**XML&Dom4j**

# 学习目标

1. 能够说出 XML 的作用
2. 了解 XML 的组成元素
3. 能够说出有哪些 XML约束技术
4. 能够说出解析 XML 文档 DOM方式原理
5. 能够说出Dom4j常用的类
6. 能够通过Dom4j得到文档对象
7. 能够读取Dom树上的元素对象
8. 能够使用 XPath 解析 XML 文档

# XML介绍

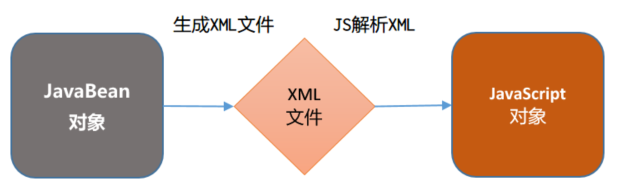
## 什么是XML

1. 什么是HTML：Hyper Text Markup Language 超文本标记语言
2. 英文：eXtensible Markup Language 可扩展标记语言
   1. 可扩展：所有的标记都是自定义的，由程序员自己去扩展。
   2. 标记语言：整个文件由标记组成，类似于HTML。不同于编程语言

## XML作用

XML在企业开发中主要有两种应用场景：

1. 数据交换：可以用于不同的语言或不同的系统之间进行数据的交换。



1. 配置文件：用于框架或工具的配置文件，这也是我们后期主要使用到的功能。

<default-config>

<!--数据库连接配置-->

<!--驱动-->

<property name="driverClass">com.mysql.jdbc.Driver</property>

<!--连接字符串-->

<property name="jdbcUrl">jdbc:mysql://localhost:3306/day19</property>

<!--用户名-->

<property name="user">root</property>

<!--密码-->

<property name="password">root</property>

<!--与连接池相关的配置-->

<!--初始连接数-->

<property name="initialPoolSize">5</property>

<!--最大连接数-->

<property name="maxPoolSize">10</property>

<!--等待多久以后超时-->

<property name="checkoutTimeout">3000</property>

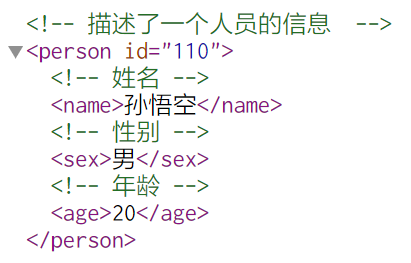
</default-config>

## XML 与 HTML 的主要差异

|  |  |  |
| --- | --- | --- |
| 区别 | HTML超文本标记语言 | XML可扩展标记语言 |
| 功能 | 用于表示层，直接面向用户。用于数据展示 | 用于数据访问层，用于数据的存储。 |
| 大小写 | 不区分大小写 | 区分大小写 |
| 语法严谨 | 不严谨，有错误浏览器也可以解析 | 非常严谨，只要有错误，解析器就不能解析 |
| 可扩展性 | 所有标签都有自己的功能，而且是固定好的 | 可以扩展的，所有的标签都是程序员自己创建出来。 |

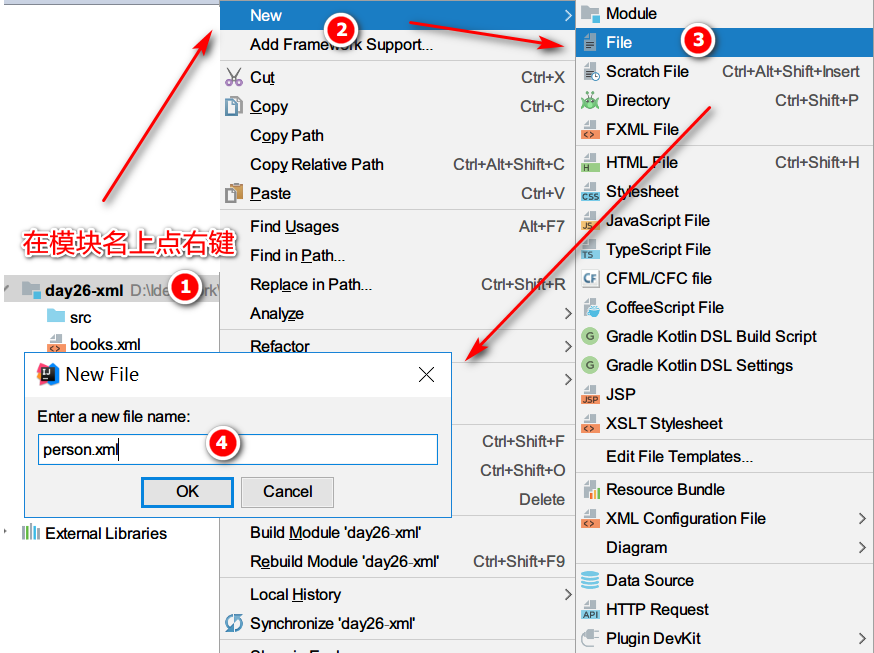
## 案例：XML文件的编写

* 需求：编写xml文档，用于描述人员信息，person代表一个人员，id是人员的属性代表人员编号。人员信息包括age年龄、name姓名、sex性别信息。
* 效果：使用浏览器查看person.xml文件效果如下：



* 实现步骤

步骤1：使用idea开发工具，选择当前项目鼠标右键新建“”，如图



步骤2：编写文件person.xml文件，内容如下：

<!--描述了一个人员的信息 -->

<person id="110">

<!--姓名-->

<name>孙悟空</name>

<!--性别-->

<sex>男</sex>

<!--年龄-->

<age>20</age>

</person>

步骤3：如图点击浏览器运行



# XML的组成

XML文件一共由7个组成元素构成：

1. 文档声明 Document Declare
2. 元素 Element
3. 属性 Attribute
4. 注释 Comment
5. 实体字符 Entity Character
6. 字符数据区 Character Data
7. 处理指令 Process Instruction

## 文档声明

|  |
| --- |
| 声明：<?xml version="1.0" encoding="utf-8"?> |
| 格式： <?xml 开头 ?>结尾 |
| 位置：必须出现在第一行第一列 |

### 声明的三个属性

|  |  |
| --- | --- |
| 文档声明的三个属性 | 说明 |
| version | 使用的XML版本，我们取值为1.0 |
| encoding | XML使用字符集，如果省略，默认是utf-8 |
| standalone | yes/no 很少使用，说明当前的XML文件是否是一个独立的文件，默认是yes |

### 版本说明：

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

## 元素Element

|  |
| --- |
| 如：<person> 就是元素，又称为标签 |
| 语法：< 开头 >结尾 中间部分就是元素名字，可以有一个或多个属性 |
| 主体部分：分为有主体和没有主体的标签，如果没有主体的标签，必须使用结束标记，如：<br/> |
| 大小写区分 |
| 命名不能有空格，不能有冒号 |
| 每个文档只能有一个根元素 |

## 属性Attribute

|  |
| --- |
| 如：<person id="110"> 其中id就是一个属性 |
| 属性必须出现在开始标签 |
| 必须使用引号引起来，既可使用双引号也可以使用单引号 |
| 一个标签可以有多个属性，但不能出现同名的属性 |
| 属性的命名不能有空格和冒号 |

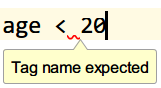
## 注释Comment

|  |
| --- |
| <!--注释内容--> |
| 与HTML的注释相同，但注释不能嵌套 |

## 实体字符 Entity Character

XML中的实体字符与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实体字符。比HTML更加严谨。

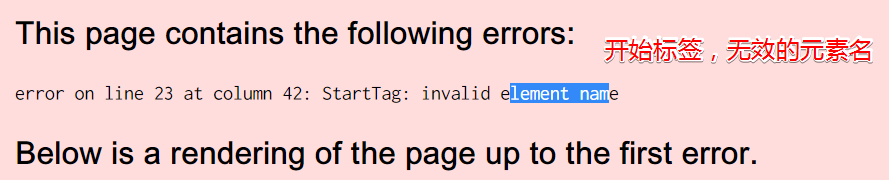
* 语法：以&开头，以;结尾



|  |  |  |
| --- | --- | --- |
| 说明 | 字符 | 转义字符 |
| 小于 | < | &lt; |
| 大于 | > | &gt; |
| 双引号 | " | &quot; |
| 单引号 | ' | &apos; |
| 与符号 | & | &amp; |

### 实体字符应用示例：

* 需求：有一个<sql>标签，中间包含sql语句，查询所有年龄小于20岁的学生。



* 错误的写法：

<sql>

select \* from student where age < 20

</sql>

* 正确的写法：

<sql>

select \* from student where age &lt; 20

</sql>

## 字符数据区

为什么要使用字符数据区： 如果出现了大量需要转义的字符，又不想使用实体字符，可以使用字符数据区CDATA，字符数据区中的数据不由解析器来解析，都做为纯文本来使用。

|  |
| --- |
| <![CDATA[文本数据]]> |
| 不由 XML 解析器进行解析的纯文本数据(Unparsed Character Data) |
| 格式：<![CDATA[ 开头 ]]>结尾 |
| 不能嵌套 |

<sql>

<![CDATA[

select \* from student where age < 20

]]>

</sql>

## 处理指令

|  |
| --- |
| <?xml-stylesheet type="text/css" href="文件名.css"?> |
| 可以对XML文件设置css样式 |
| 格式： <?xml-stylesheet 开头 以?>结尾 |
| 作用：处理指令，简称PI(Processing instruction)用来指挥解析引擎如何解析XML文档内容。 |

### 处理指令示例：

* 现有person.xml文档内容如下：

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

<persons>

<!--描述了一个人员的信息 -->

<person id="110">

<!--姓名-->

<name>孙悟空</name>

<!--性别-->

<sex>男</sex>

<!--年龄-->

<age>20</age>

</person>

<!--描述了一个人员的信息 -->

<person id="120">

<!--姓名-->

<name>猪八戒</name>

<!--性别-->

<sex>男</sex>

<!--年龄-->

<age>18</age>

</person>

<!--写一个SQL语句-->

<sql>

select \* from student where age &lt; 20

</sql>

<sql>

<![CDATA[

select \* from student where age < 20

]]>

</sql>

</persons>

* 有样式文件person.css文件，内容如下：

name {

color: red;

font-size: 20px;

}

sex {

color: green;

}

age {

color: blue;

}

* 操作person.xml文档应用person.css样式文件：

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

* 使用浏览器浏览效果如下：



## 一个格式良好的XML有以下特点

1. 必须以XML声明开头
2. 必须拥有唯一的根元素
3. 开始标签必须与结束标签相匹配
4. 元素对大小写敏感
5. 所有的元素必须关闭
6. 所有的元素必须正确地嵌套
7. 特殊字符必须使用实体字符

# XML文件的约束

XML文件约束作用：用来约束XML的数据，一个XML文件中能够出现哪些标签。每个标签中能够出现哪些子标签，每个标签能够有哪些属性。每个标签的主体内容可以出现哪些。

|  |
| --- |
| 两种XML约束 |
| DTD约束 |
| Schema约束 |

## DTD约束

### 概念

* DTD：文档类型定义 Document Type Definition
* 作用： 用来约束XML文档，保证XML文件正确性和数据的有效性。

### DTD的使用案例:

步骤1：新建book.dtd文件，选择项目鼠标右键“NEW->File",文件名为“book.dtd”

步骤2：book.dtd文件内容如下

<!ELEMENT 书架 (书+)>

<!ELEMENT 书 (书名,作者,售价)>

<!ELEMENT 书名 (#PCDATA)>

<!ELEMENT 作者 (#PCDATA)>

<!ELEMENT 售价 (#PCDATA)>

步骤3：新建books.xml，代码如下

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

<!--引入dtd约束文件-->

<!DOCTYPE 书架 SYSTEM "book.dtd">

<书架>

<书>

<书名>人鬼情喂鸟</书名>

<作者>保罗.克林吨</作者>

<售价>300</售价>

</书>

<书>

<书名>JavaEESSH</书名>

<作者>NewBoy</作者>

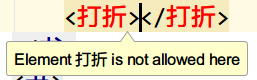
<售价>38</售价>

</书>

</书架>

|  |  |
| --- | --- |
| 导入DTD文件的两种格式 | 说明 |
| <!DOCTYPE 根元素 SYSTEM "DTD文件"> | 表示引用系统本地的DTD文件，使用范围比较窄，通常用于个人或公司或团队。 |
| <!DOCTYPE 根元素 PUBLIC "文件描述" "DTD文件"> | 表示公共DTD约束文件，通常在互联网上广泛使用的，后期使用这种比较多的。 |

步骤4：idea开发工具books.xml的dtd约束验证不通过的效果如下



### DTD学习要求

在企业实际开发中，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。

## Schema约束

### 为什么使用Schema

1. 
2. 因为DTD是一个普通的文本文件，如果DTD本身有错误，也不容易查找出来。

|  |
| --- |
| Schema概念 |
| Schema：是一种用来代替DTD约束的新的方式，功能上比DTD更加强大，可以验证数据类型。本身还是一个XML文件。 |
| 扩展名：xsd —— XML Schema Document |

### Schema的演示案例

* 实现步骤

步骤1：新建schema约束文件bookshelf.xsd，对售价约束数据类型，代码如下：

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.itcast.cn"

elementFormDefault="qualified">

<xs:element name='书架'>

<xs:complexType>

<xs:sequence maxOccurs='unbounded'>

<xs:element name='书'>

<xs:complexType>

<xs:sequence>

<xs:element name='书名' type='xs:string'/>

<xs:element name='作者' type='xs:string'/>

<xs:element name='售价' type='xs:double'/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

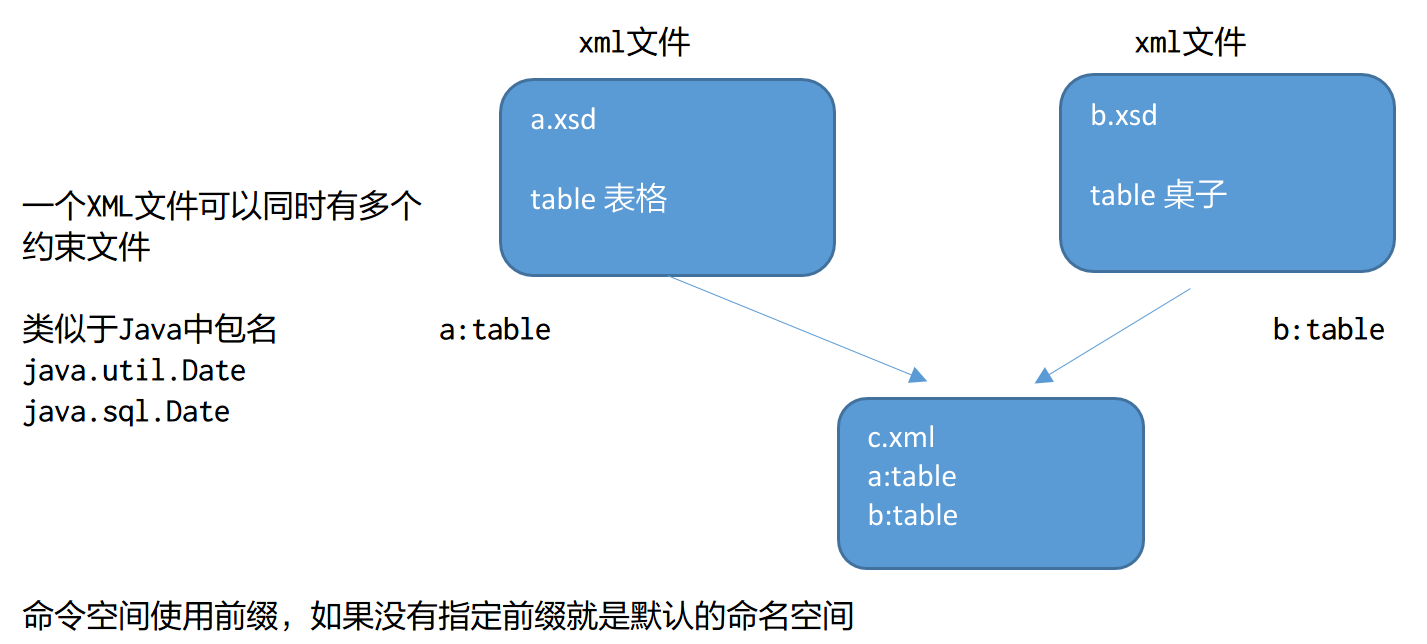
</xs:element>

</xs:schema>

* targetNamespace表示命令空间，我们使用xsd约束的时候，用来引入到XML文件中。

### 什么是命名空间：

1. 如果有多个约束文件中有相同的标签，使用不同的前缀来区分，引入不同的xsd中约束。
2. 类似于Java中包
3. 如果没有前缀的约束，称为默认命名空间



步骤2：新建books2.xml使用schema约束文件bookshelf.xsd，代码如下

* 语法：

|  |
| --- |
| <根元素 xmlns="命名空间" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="命名空间 xsd约束文件名">  </根元素> |

* 代码：

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

<书架 xmlns="http://www.itcast.cn"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.itcast.cn bookshelf.xsd">

<书>

<书名>西游记</书名>

<作者>金正恩</作者>

<售价>45.3</售价>

</书>

<书>

<书名>东游记</书名>

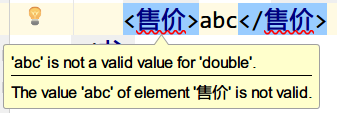
<作者>金针菇</作者>

<售价>300</售价>

</书>

</书架>

步骤3：开发工具提示效果



### Schema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。我们只需要借助开发工具，在现有的xsd约束下，写出正确的xml文件即可。

# XML解析

## 什么是XML的解析

读取XML文件中所有的元素和数据，封装成一个JavaBean，在Java中来使用这些数据。

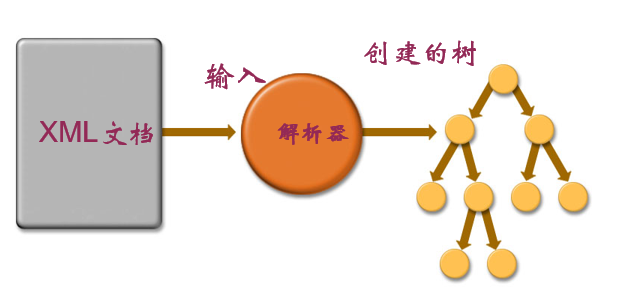
## 解析方式和解析器

### 两种解析方式：

* 1. DOM： Document Object Model 文档对象模型，会在内存中创建一棵DOM树。可以对这棵树上所有的结点进行查询，修改，删除和添加。

a）优点：可以方便的操作XML中任何一个结点，可以进行增删改查的操作。

b）缺点：如果这个XML文件很大的时候，可能会出现内存泄露的风险。



* 1. SAX：
  + a）优点：事件驱动型解析方式，读取一行解析一行，理论上可以解析任意大小的XML文件。
  + b）缺点：不能对XML文件进行修改，只能读取。读取完毕以后不能再访问之前已经访问过的元素。

### 常见的XML解析开发包DOM

|  |  |
| --- | --- |
| 开发工具包 | 说明 |
| JAXP | Oracle官方提供的API，同时支持DOM和SAX的开发 |
| JDOM | JDOM是一个开源项目，它基于树型结构，利用纯JAVA的技术对XML文档实现解析、生成、序列化以及多种操作。 |
| Dom4j | 是JDOM的升级品，用来读写XML文件的。具有性能优异、功能强大和极其易使用的特点，它的性能超过sun公司官方的dom技术，同时它也是一个开放源代码的软件，Hibernate也用它来读写配置文件。 |
| Jsoup | jsoup 是一款Java 的HTML和XML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。 |

# Dom4j的学习

## Dom4j概述

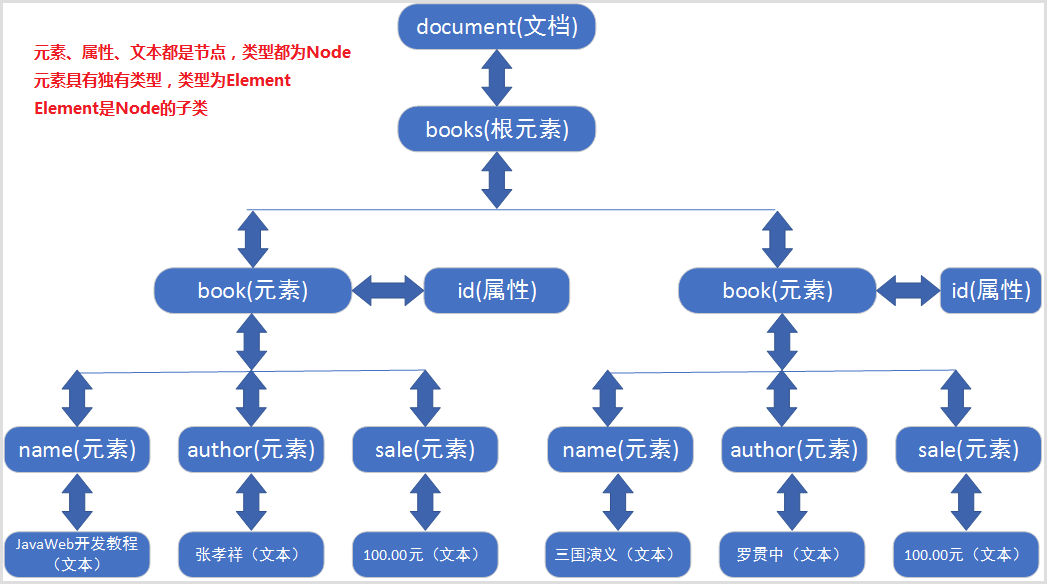
采用DOM方式来解析XML文件

### DOM解析原理及结构模型

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="0001">  
 <name>JavaWeb开发教程</name>  
 <author>张孝祥</author>  
 <sale>100.00元</sale>  
 </book>  
 <book id="0002">  
 <name>三国演义</name>  
 <author>罗贯中</author>  
 <sale>100.00元</sale>  
 </book>  
</books>

* 结构模型



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。

### DOM树的组成元素

|  |  |
| --- | --- |
| 组成 | 说明 |
| Document | 文档对象：代表整个xml文档 |
| Node | 节点 |
| Element | 元素 |
| Attribute | 属性 |
| Text | 文本 |

### Dom4j使用步骤

1. 去官网下载 zip 包。<http://www.dom4j.org/>
2. 在项目中创建一个文件夹：lib
3. 将dom4j-2.1.1.jar文件复制到 lib 文件夹
4. 在jar文件上点右键，选择 Add as Library -> 点击OK
5. 在类中导包使用

## 获得Document对象

### 概述

一个XML文件加载到内存中时就会创建一个Document对象，该Document对象包含了XML中的所有节点内容，通过该对象就可以获得对应XML文件的所有内容。

### 得到Document对象

文件Contact.xml放在src目录下

1. 创建一个SAXReader对象，用于读取 xml 文件。
2. 从类路径下加载xml文件，得到输入流对象
3. 通过 SAXReader对象的read(InputStream in )方法，从输入流中读取，生成文档对象。

package com.itheima.xml;

import org.dom4j.Document;

import org.dom4j.DocumentException;

import org.dom4j.io.SAXReader;

import org.junit.Test;

import java.io.InputStream;

/\*\*

\* 使用 dom4j 解析工具读取 xml 文件

\*/

public class Demo1 {

/\*\*

\* 得到文档对象

\*/

@Test

public void getDocument() throws DocumentException {

//创建SAXReader对象

SAXReader reader = new SAXReader();

//读取src目录下的文件，得到输入流

InputStream in = Demo1.class.getResourceAsStream("/Contact.xml");

//从输入流中得到文档对象

Document document = reader.read(in);

System.out.println(document);

}

### Document常用方法

|  |  |
| --- | --- |
| 方法名 | 功能说明 |
| Element getRootElement() | 获得根元素 |

### 示例演示

* 需求：获得xml根元素对象并输出到控制台
* 步骤：

1. 得到SAXReader对象
2. 得到文档对象
3. 得到根元素
4. 打印输出

* 代码

/\*\*

\* 得到根元素

\*/

@Test

public void getRoot() throws DocumentException {

//得到SAXReader对象

SAXReader reader = new SAXReader();

//得到文档对象

Document document = reader.read(Demo1.class.getResourceAsStream("/Contact.xml"));

//得到根元素

Element rootElement = document.getRootElement();

//打印元素

System.out.println(rootElement);

}

## Element对象

### 概述

DOM树上的每一个标签都是一个Element对象。

### Element常用方法

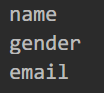
|  |  |
| --- | --- |
| 方法名 | 功能说明 |
| String getName() | 得到标签的名字 |
| Element element(name) | 获得指定名字的元素对象 |
| List<Element> elements() | 获得当前元素下所有的子元素对象 |
| List<Element> elements(name) | 获得当前元素下指定名字的所有子元素对象 |

### 案例演示

* 需求：

元素方法的使用，得到contact下所有的子元素，并且输出元素名

* 效果



* 步骤

1. 得到SAXReader对象
2. 得到Document对象
3. 得到根元素
4. 得到contact元素
5. 得到contact下所有的子元素，并输出子元素的名字

* 代码

/\*\*

\* 得到contact下所有的子元素，并输出子元素的名字

\*/

@Test

public void elementsName() throws DocumentException {

//1). 得到SAXReader对象

SAXReader reader = new SAXReader();

//2). 得到Document对象

Document document = reader.read(Demo1.class.getResourceAsStream("/Contact.xml"));

//3). 得到根元素

Element root = document.getRootElement();

//4).得到contact元素

Element contactElement = root.element("contact");

//5) 得到contact下所有的子元素，并输出子元素的名字

List<Element> elementList = contactElement.elements();

for (Element element : elementList) {

System.out.println(element.getName());

}

}

## Attribute对象

### 概述

DOM树上每一个元素的属性都是一个Attribute对象。获得Attribute对象前提：先获得元素Element对象，再调用Element对象与属性相关的方法。

### Element中与属性相关方法

|  |  |
| --- | --- |
| Element中的方法 | 功能说明 |
| Attribute attribute(name) | 根据属性名获得一个属性对象 |
| String attributeValue(name) | 根据属性名获得属性值，返回字符串 |
| List<Attribute> attributes() | 获得当前元素的所有属性对象，返回List集合 |

### Attribute常用方法

|  |  |
| --- | --- |
| 方法名 | 功能说明 |
| String getName() | 获得属性的名字 |
| String getValue() | 获得属性的值 |

### 结论：得到属性值的两种方式

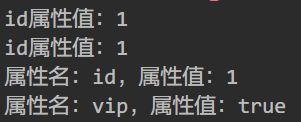
1. 先得到Attribute对象，再通过对象的方法得到属性名和属性值
2. 直接通过属性的名字得到字符串类型的属性值。

### 案例演示

* 需求

1. 使用2种方式，得到contact上id属性值
2. 在contact元素上添加一个vip的属性值为true/false，得到contact上所有的属性名和属性值

* 效果



* 步骤

1. 创建SAXReader
2. 调用read 方法，读取 xml 文件
3. 得到根元素
4. 获得第1个contact元素对象
5. 通过方式1：得到contact上id属性值
6. 通过方式2：得到contact上id属性值
7. 得到contact上所有的属性名和属性值

* 代码

/\*\*

\* 属性方法的使用

\*/

@Test

public void attribute() throws DocumentException {

//创建SAXReader

SAXReader reader = new SAXReader();

//调用read 方法，读取 xml 文件

Document document = reader.read(Demo1.class.getResourceAsStream("/Contact.xml"));

//得到根元素

Element rootElement = document.getRootElement();

//获得第1个contact元素对象

Element contactElement = rootElement.element("contact");

//通过方式1：得到contact上id属性值

String id = contactElement.attributeValue("id");

System.out.println("id属性值：" + id);

//通过方式2：得到contact上id属性值

Attribute idAttr = contactElement.attribute("id");

String value = idAttr.getValue();

System.out.println("id属性值：" + value);

//得到contact上所有的属性名和属性值

List<Attribute> attributes = contactElement.attributes();

for (Attribute attribute : attributes) {

System.out.println("属性名：" + attribute.getName() + "，属性值：" + attribute.getValue());

}

}

## Text文本元素

### 概述：

得到文本元素的前提：先得到元素Element，再得到元素下的文本

* + 注：空格、换行、制表符：也是属于文本的一部分，所以在解析xml文件的时候，格式化XML文件要注意。

### 得到文本信息的方法：

|  |  |
| --- | --- |
| Element元素中的方法 | 说明 |
| String getText() | 得到当前元素内部的文本 |
| String elementText(元素名) | 通过子元素名字得到子元素内部的文本 |
| String elementTextTrim(元素名) | 通过子元素名字得到子元素内部的文本，并且去掉前后的空格 |

### 案例：对文本的操作

* 需求：

得到第1个contact中的name元素，输出name元素的文本。分别通过下面三个方法得到：

1. 通过getText()方法得到
2. 直接得到元素的内容
3. 得到去掉空格的文本

* 效果



* 步骤

1. 得到Document对象
2. 得到根元素
3. 得到contact标签
4. 得到name标签中的文本
5. 得到name标签
6. 再得到name中的文本

* 代码

/\*\*

\* 文本内容的获取

\*/

@Test

public void text() throws DocumentException {

//1. 得到Document对象

Document document = new SAXReader().read(Demo1.class.getResourceAsStream("/Contact.xml"));

//2. 得到根元素

Element rootElement = document.getRootElement();

//3. 得到contact标签

Element contactElement = rootElement.element("contact");

//4. 得到name标签中的文本

System.out.println(contactElement.elementText("name"));

//5. 得到name标签

Element nameElement = contactElement.element("name");

//6. 再得到name中的文本

System.out.println(nameElement.getText());

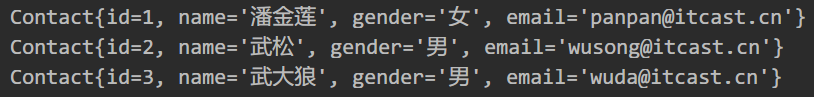
}

## XML解析案例

### 需求说明

利用 Dom4j 的知识，将 Contact.xml 文件中的联系人数据封装成集合，其中每个元素是实体类 Contact。打印输出 List 中的每个元素

### 运行效果



### 数据准备

* Contact.xml

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

<contactList>

<contact id="1">

<name>潘金莲</name>

<gender>女</gender>

<email>panpan@itcast.cn</email>

</contact>

<contact id="2">

<name>武松</name>

<gender>男</gender>

<email>wusong@itcast.cn</email>

</contact>

<contact id="3">

<name>武大狼</name>

<gender>男</gender>

<email>wuda@itcast.cn</email>

</contact>

</contactList>

* Contact.java实体类

package com.itheima.entity;

/\*\*

\* 联系人实体类

\*/

public class Contact {

private int id;

private String name;

private String gender;

private String email;

@Override

public String toString() {

return "Contact{" +

"id=" + id +

", name='" + name + '\'' +

", gender='" + gender + '\'' +

", email='" + email + '\'' +

'}';

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

### 开发步骤

1. 创建集合对象用于封装所有的联系人
2. 获得文档对象
3. 获取根元素
4. 获得所有的contact元素
5. 遍历contact元素集合
6. 每个contact创建一个联系人对象
7. 获得id属性值，封装到id属性中
8. 得到contact下的子元素的文本封装其它的属性
9. 将联系人对象添加到集合中
10. 遍历输出每一个联系人对象

### 代码实现

package com.itheima.xml;

import com.itheima.entity.Contact;

import org.dom4j.Document;

import org.dom4j.DocumentException;

import org.dom4j.Element;

import org.dom4j.io.SAXReader;

import java.util.ArrayList;

import java.util.List;

public class Demo2Contact {

public static void main(String[] args) throws DocumentException {

//1)创建集合对象用于封装所有的联系人

List<Contact> contactList = new ArrayList<>();

//2)获得文档对象

Document document = new SAXReader().read(Demo2Contact.class.getResourceAsStream("/Contact.xml"));

//3)获取根元素

Element rootElement = document.getRootElement();

//4)获得所有的contact元素

List<Element> contactElements = rootElement.elements("contact");

//5)遍历contact元素集合

for (Element contactElement : contactElements) {

//6)每个contact创建一个联系人对象

Contact contact = new Contact();

//7)获得id属性值，封装到id属性中

contact.setId(Integer.parseInt(contactElement.attributeValue("id")));

//8)得到contact下的子元素的文本封装其它的属性

contact.setName(contactElement.elementText("name"));

contact.setGender(contactElement.elementText("gender"));

contact.setEmail(contactElement.elementText("email"));

//9)将联系人对象添加到集合中

contactList.add(contact);

}

//10)遍历输出每一个联系人对象

for (Contact contact : contactList) {

System.out.println(contact);

}

}

}

# XPath表达式

## Xpath概述

XPath即为[XML](https://baike.baidu.com/item/XML)路径语言：XML Path Language

XPath使用路径表达式来选取HTML或XML文档中的元素节点或属性节点。

### Xpath表达式分类

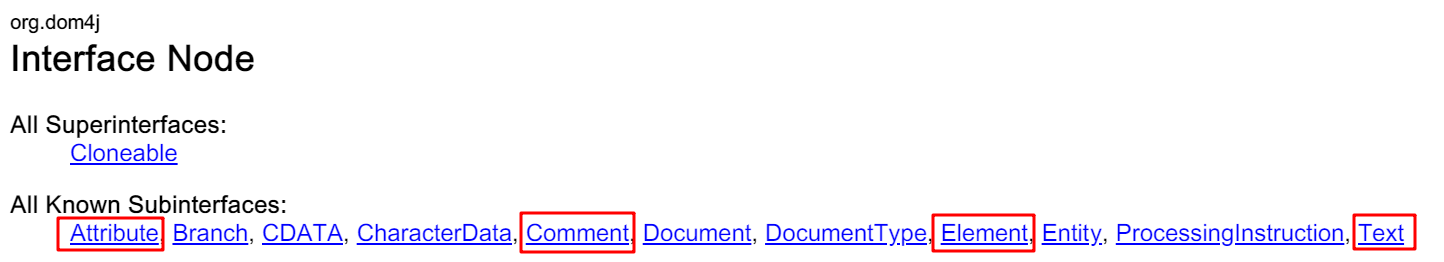
1. 绝对路径表达式
2. 相对路径表达式
3. 全文搜索表达式
4. 属性查找表达式

## dom4j与Xpath相关的方法

|  |  |
| --- | --- |
| Node节点的方法 | 功能说明 |
| Node selectSingleNode(String xpath) | 根据路径查找节点对象，如果有多个返回第一个 |
| List selectNodes(String xpath) | 根据路径查找节点对象，返回集合 |

### Node对象

DOM树中每个元素都抽象成一个节点对象，节点是所有元素的父元素。



* + 注：使用XPath必须导入: jaxen-1.1.2.jar包

## 绝对路径表达式

### 语法

|  |  |
| --- | --- |
| 语法 | 说明 |
| /元素/子元素/孙元素 | / 代表XML文档对象，表示一级一级往下查找 |

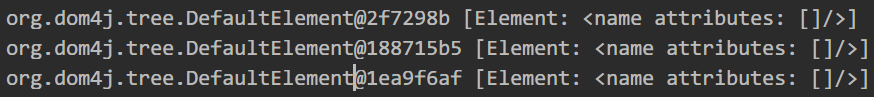
### 示例

使用JUnit，在@before方法中创建一个全局文档对象即可

* 需求：

采用绝对路径获取从根节点开始逐层的查找/contactList/contact/name/ 节点列表并打印信息

* 效果



* 步骤

1. 创建 XML 解析器对象
2. 读取 XML 文档并获得Document对象
3. 定义 Xpath 表达式：/contactList/contact/name
4. 调用Document对象的selectNodes()方法执行Xpath获得节点集合
5. 遍历输出每个节点

* 代码

package com.itheima.xml;

import org.dom4j.Document;

import org.dom4j.DocumentException;

import org.dom4j.Node;

import org.dom4j.io.SAXReader;

import org.junit.Before;

import org.junit.Test;

import java.util.List;

/\*\*

\* XPath的使用

\*/

public class Demo3XPath {

private Document document;

//在每个测试方法前运行：得到文档对象

@Before

public void init() throws DocumentException {

document = new SAXReader().read(Demo3XPath.class.getResourceAsStream("/Contact.xml"));

}

/\*\*

\* 绝对路径

\*/

@Test

public void absolutePath() {

//定义 Xpath 表达式

String xpath = "/contactList/contact/name";

//执行Xpath获得节点集合

List<Node> nodeList = document.selectNodes(xpath);

for (Node node : nodeList) {

System.out.println(node);

}

}

}

## 相对路径表达式

### 语法

|  |  |
| --- | --- |
| 格式 | 说明 |
| ./子元素/孙元素 | 相对当前节点元素位置下查找 |

### 示例

* 需求：

先采用绝对路径获取 contactList 节点，再采用相对路径获取下一级contact 节点的name子节点并打印信息。

* 步骤

1. 创建 XML 解析器对象
2. 读取 XML 文档并获得Document对象
3. 定义 Xpath 表达式：/contactList
4. 调用Document对象的 selectSingleNode 方法执行Xpath获得根节点对象
5. 通过根节点对象调用selectNodes方法执行相对路径表达式：./contact/name
6. 打印输出所有的节点

* 代码

// 相对路径表达式

@Test

public void relativePath()throws Exception{

// 绝对路径表达式

String xpath = "/contactList";

// 使用Xpath查找节点

Element rootElement = (Element) document.selectSingleNode(xpath);

// 相对路径表达式：查询rootElement节点下所有name子节点

List<Node> contactNodes = rootElement.selectNodes("./contact/name");

for (Node node : contactNodes) {

System.out.println(node);

}

}

## 全文搜索表达式

### 语法

|  |  |
| --- | --- |
| 语法 | 说明 |
| //元素 | // 表示直接选取对应的节点，无论在哪一级 |

### 语法举例

|  |  |
| --- | --- |
| 举例 | 说明 |
| //contact | 查找XML文档中的所有contact节点 |
| //contact/name | 查询所有contact节点下的name子节点 |
| //contact//name | 查询所有contact节点下的name子孙节点 |

### 示例

* 需求

直接全文搜索所有的 name元素并打印

* 步骤

1. 创建Xpath表达式 //name
2. 使用selectNodes()方法查询所有的name节点

* 代码

// 全文搜索表达式

@Test

public void globalSearch() throws DocumentException {

// Xpath表达式: // 表示全文搜索，不管在哪一级

String xpath = "//name";

// 使用Xpath查找节点

List<Node> nodes = document.selectNodes(xpath);

for (Node node : nodes) {

System.out.println(node);

}

}

## 属性查找表达式

### 语法

|  |  |
| --- | --- |
| 格式 | 说明 |
| //@属性名 | 获得指定属性名的所有属性节点 |
| //元素[@属性名] | 获取元素包含"属性名"的所有元素 |
| //元素[@属性名='值'] | 获取元素包含"属性名=值"的所有元素 |

### 示例

* 需求：

1. 查找所有id属性节点
2. 查找包括id属性的contact元素
3. 查找包括id属性且属性名为的contact元素

* 步骤

1. 创建Xpath表达式
2. 使用selectNodes()方法查询所有的节点

* 代码

// 属性查找

@Test

public void attributeFind() throws Exception {

// 查找所有id属性节点

//String xpath = "//@id";

// 查找包括id属性的contact元素

// String xpath = "//contact[@id]";

// 查找包括id属性且属性名为S100的contact元素

String xpath = "//contact[@id='3']";

// 使用Xpath查找节点

List<Node> nodes = document.selectNodes(xpath);

for (Node node : nodes) {

System.out.println(node);

}

}