# 第11天【XML：可扩展标记语言】

## 主要内容

1. XML定义
2. XML文档结构
3. XML优势
4. XML作用
5. DTD约束
6. XML Schema约束
7. XML解析技术
8. 使用DOM解析XML
9. 使用DOM4J解析XML
10. 使用DOM4J操作XML

## 学习目标

|  |  |
| --- | --- |
| 知识点 | 要求 |
| XML定义 | 理解 |
| XML文档结构 | 理解 |
| XML优势 | 了解 |
| XML作用 | 了解 |
| DTD约束 | 了解 |
| XML Schema约束 | 了解 |
| XML解析技术 | 了解 |
| 使用DOM解析XML | 了解 |
| 使用DOM4J解析XML | 了解 |
| 使用DOM4J操作XML | 了解 |

## 一、XML简介

### 1.1 XML定义

XML，Extensible Markup Language（可扩展标记语言），由SGML语言发展而来，允许用户自定义标签，可以将标签和内容有效分离。它逐渐演变为一种跨平台的数据交换格式（一种在不同平台、不同系统之间的数据交换格式），一种轻量级的持久化方案（保存简单数据，无需使用数据库）。

XML只是纯文本而已，只是一种独立于软件、硬件的数据存储和传输工具。它可对外提供一下信息，但于C、Java这些编程语言不同，XML无法提供任何“动态行为”。

和HTML提供预定义标签不同，开发者可以自定义任意标签，因此具有很强的扩张性；不同于HTML侧重于数据的展示，XML更加关注数据的存储和传输；不同于HTML可以使用浏览器来解析并显示，XML需自行编写软件或程序，才能传送、接收、显示这个文档。

1998年2月10日，XML正式成为W3C的推荐标准（晚于HTML）。XML和JavaEE是两种不同的技术，但是其广泛的用在JavaEE开发的各个方面，比如使用XML作为配置文件，使用XML进行数据交换。

### 1.2 XML文档结构

#### 【示例1】XML文档示例

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <!-- 学生信息 --> <**students**>  <**student id="001"**>  <**name**>李明</**name**>  <**age**>23</**age**>  <**score**>98</**score**>  </**student**>  <**student id="002"**>  <**name**>李刚</**name**>  <**age**>23</**age**>  <**score**>88</**score**>  </**student**> </**students**> |

**一个XML文件分为如下几部分内容：**

1. 文档声明

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

XML文档总是以XML声明开始，定义了XML的版本信息和所使用的的编码等信息。

1. 元素

<**name**>称为开始标签、</**name**>称为结束标签，“李明”表示标签的内容。开始标签、内容、结束标签组合成元素。元素是XML文档的主要部分，元素内容可以是普通文本，也可以是子元素。比如**student**元素的内容就是多个子元素。

一个XML文档有且仅有一个根元素，比如**students**。

1. 属性

元素<**student id="001"**>中的id就是属性名，001是属性值，属性值要使用双引号括起来。属性加载一个元素的开始标签上，用来对元素进行描述。一个元素可以有多个属性，空格隔开。属性没有先后顺序，同一个XML元素不允许同名属性。

1. 注释

<!-- 学生信息 -->

对XML内容进行解释说明的文字

1. CDATA标记、字符实体

有时元素文本中会一些特殊字符，比如<、>、”、&等，这些字符在XML文档结构本身中已经用到了，此时主要通过两种办法，实现正确解析这些特殊字符。

方法1：个别的特殊字符可以使用字符实体替换

|  |  |  |
| --- | --- | --- |
| 字符实体 | 特殊字符 | 含义 |
| &lt; | < | 小于 |
| &gt; | > | 大于 |
| &amp; | & | 和号 |
| &apos; | ' | 单引号 |
| &quot; | " | 双引号 |

严格地讲，在 XML 中仅有字符 "<"和"&" 是非法的。单引号、双引号和大于号是合法的，但是把它们替换为实体引用是个好的习惯。

方法2：大量的特殊字符可以CDATA标记来处理

CDATA标记中的所有字符都会被当做普通字符来处理，而不是XML标签。

定义CDATA标记的语法

**<![CDATA[**

**要显示的字符**

**]]>**

#### 【示例2】XML的CDATA标记

|  |
| --- |
| <**student id="002"**>   <**name**>李刚</**name**>  <**age**><![CDATA[<young>10</young>]]></**age**>  <**score**>**&lt;**100</**score**> </**student**> |

**格式良好的 XML 文档：遵循XML文档的基本规则**

* 元素正确嵌套
* XML文件的第一行必须是xml声明
* XML文件只能有一个根节点
* 英文字符的大小写是有差异的
* 开始的控制标记与结束的控制标记缺一不可
* 属性值的设置必须被""包围起来

**有效的XML文档**

* 首先必须是格式良好的（语法约束）
* 使用DTD和XSD(XML Schema)定义语义约束

|  |
| --- |
| **新手雷区**  XML可以任意定义标签、任意嵌套，任意定义属性不假，但是也必须保证XML的接收者能准确的理解信息才行。所以就需要双方约定一套规则，创建这按照规则保存信息，接收者按照规则读取信息。这些规则称为语义约束。这些规则主要包括：   * 文档中根元素是什么 * 文档中允许哪些元素 * 元素允许哪些子元素，顺序是否有要求，是否可以重复出现 * 元素允许哪些属性，属性的类型、属性的取值范围、默认值等   可以把语义约束理解为XML文件的一个模板。对于不同的XML文件可以有不同的模板 |

### 1.3 XML优势

* 简单性

遵循XML文档的基本规则的前提下，可以任意自定义标签

* 良好的可读性

遵循XML文档的基本规则的前提下，标签名见名知义，具有良好的嵌套关系，带有了 良好的可读性。

* 可扩展性

可以根据XML的基本语法来进一步限定使用范围和文档格式，从而定义一种新的语言

* 可以轻松的跨平台应用

XML文档是基于文本的，所以很容易被人和机器阅读，也非常容易使用，便于不同设 备和不同系统间的信息交换

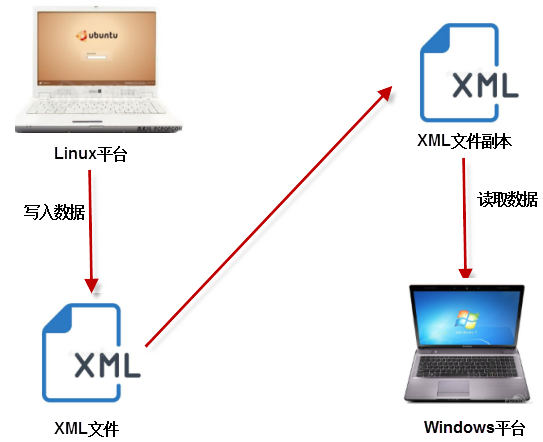
* 数据内容与显示的分离

在XML文档中，数据的显示样式已从文档中分离出来，而放入相关的样式表文件中。 这样一来如果要改动数据的表现形式，就不需要改动数据本身，而只要改动控制数据显 示的样式表文件就可以了。

### 1.4 XML作用

* 数据交换

由于各个计算机所使用的操作系统、数据库不同，因此数据之间的交换向来是件头痛的事，可以使用XML来交换数据。比如可以通过XML实现Linux和Windows平台之间的数据传输。这个过程中先将Linux平台中数据通过程序保存到xml文件中，再在Windows中通过程序读取xml中的数据。



* 数据配置

使用XML配制文件可读性强，灵活性高。在后面JavaEE的Servlet、Filter、Listener、JavaEE开源框架的Spring、SpringMVC、MyBatis开发中会经常使用XML存储配置信息。

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   xmlns="http://java.sun.com/xml/ns/javaee"   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee   http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"   version="3.0"**>  <**welcome-file-list**>  <**welcome-file**>index.html</**welcome-file**>  <**welcome-file**>index.htm</**welcome-file**>  <**welcome-file**>index.jsp</**welcome-file**>  </**welcome-file-list**>  <**servlet**>  <**servlet-name**>HelloServlet</**servlet-name**>  <**servlet-class**>com.bjsxt.servlet.HelloServlet</**servlet-class**>  </**servlet**>  <**servlet-mapping**>  <**servlet-name**>HelloServlet</**servlet-name**>  <**url-pattern**>/servlet/HelloServlet</**url-pattern**>  </**servlet-mapping**> </**web-app**> |

* 数据存储

数据库，比如Oracle、MySQL等提供了强有力的数据存储能力和处理能力，XML也可以用来存储数据。XML文件可以做小型数据库，也是不错的选择，我们程序中可能用到一些经常要人工配置的数据，如果放在数据库中读取不合适（因为这会增加维护数据库的工作），则可以考虑直接用XML来做小型数据库。这种方式直接读取文件显然要比读数据库快。比如MSN中保存用户聊天记录就是用XML文件。

### 本节作业

1. 使用XML描述下表中的商品信息。XML文件为product.xml

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | 名称 | 单价 | 颜色 | 尺寸 | 库存 |
| P001 | 蜘蛛王皮鞋 | 268 | 黑色 | 42 | 500 |
| P002 | ThinkPad x240 | 5678 | 黑色 | 12 | 50 |
| P003 | WD移动硬盘 | 568 | 蓝色 | 5 | 1000 |

1. 格式良好的 XML 文档和有效的XML文档的含义
2. XML的优势和作用

## 二、XML语义约束

上面已经讲解了XML语义约束的原因和必要性。实际中XML语义约束主要包括DTD和XML Schema两种约束。其中DTD是早期的语义约束，XML Schema是DTD的替代者，本身也是也个XML文件，功能也更加强大。

### 2.1 DTD约束

DTD，Document Type Definition，文档类型定义，保证XML文档格式正确性。使用DTD定义了合法的语义约束后，必须让XML文档引入该语义约束，才会生效。在XML文档中引入DTD主要包括3中方式。

* 内部DTD
* 外部DTD
* 公用DTD（引入网络中DTD）

#### 1 内部DTD

所谓内部DTD是指DTD和XML数据在同一个XML文件中。DTD定义在XML声明和XML主体内容之间。以**<!DOCTYPE根元素[** 开始，**以]>**结束。

#### 【示例3】内部DTD

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE students**[  <!ELEMENT students (student+)>  <!ELEMENT student (name,age,score)>  <!ATTLIST student id CDATA #REQUIRED>  <!ELEMENT name (#PCDATA)>  <!ELEMENT age (#PCDATA)>  <!ELEMENT score (#PCDATA)>  ]***>*** <**students**>  <**student id="001"**>  <**name**>李明</**name**>  <**age**>23</**age**>  <**score**>98</**score**>  </**student**>  <**student id="002"**>  <**name**>李刚</**name**>  <**age**>23</**age**>  <**score**>88</**score**>  </**student**> </**students**> |

|  |  |
| --- | --- |
| **DTD约束** | 含义 |
| **<!DOCTYPE students**[ | DTD约束开始，XML文件根元素是students |
| <!ELEMENT students (student+)> | students元素的子元素是student，至少一个 |
| <!ELEMENT student (name,age,score)> | student元素的子元素是name，age，score，有且仅有一个，并且必须按照固定顺序出现 |
| <!ATTLIST student id CDATA #REQUIRED> | student元素的属性列表，只有一个id，只能是字符串类型CDATA，必选，也必须指定值 |
| <!ELEMENT name (#PCDATA)> | name元素是字符串类型#PCDATA（不能再有子元素，也不能为空） |
| <!ELEMENT age (#PCDATA)> | age元素是字符串类型#PCDATA（不能再有子元素，也不能为空） |
| <!ELEMENT score (#PCDATA)> | score元素是字符串类型#PCDATA（不能再有子元素，也不能为空） |
| ]***>*** | DTD定义结束 |

注意

1. 子元素有序使用逗号：<!ELEMENT student (name,age,score)>
2. 子元素互斥使用竖线：<!ELEMENT student (name|age|score)>
3. 子元素无序没有特殊语法，变通解决：<!ELEMENT student (name|age|score)+>
4. 元素类型：字符串类型#PCDATA、空内容EMPTY、任意内容ANY、子元素
5. 子元素出现的频率

* ？表示子元素出现0次到1次 （最多出现一次）
* +表示子元素至少出现一次 （至少出现一次 ）
* \*表示子元素可以出现0到多次 （任意）

1. PCDATA：parsed character data 被解析的字符串类型 在DTD定义来指定元素类型

CDATA character data 不被解析的字符串类型 在DTD定义中指定属性类型

1. ATTLIST只能定义一个属性，如果一个元素包括多个属性，需通过多个ATTLIST定义

#### 2 外部DTD

如果不同的XML文件使用相同的DTD验证规则，如果采用内部DTD就会导致代码的重复，不利于后期修改维护。此时可以定义外部DTD，让XML文件来引入外部DTD。

#### 【示例4】外部DTD

|  |
| --- |
| **<!ELEMENT students** (**student**+)> **<!ELEMENT student** (**name**,**age**,**score**)> **<!ATTLIST student id CDATA** #REQUIRED> **<!ELEMENT name** (#PCDATA)> **<!ELEMENT age** (#PCDATA)> **<!ELEMENT score** (#PCDATA)> |

引入外部DTD的语法是

**<!DOCTYPE 根元素 SYSTEM "外部DTD文件路径"*>***

#### 【示例5】引入外部DTD

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* **<!DOCTYPE students SYSTEM "student.dtd"*>*** <**students**>  <**student id="001"**>  <**name**>李明</**name**>  <**age**>23</**age**>  <**score**>98</**score**>  </**student**>  <**student id="002"**>  <**name**>李刚</**name**>  <**age**>23</**age**>  <**score**>88</**score**>  </**student**> </**students**> |

#### 3 公用DTD

其实也是一种外部DTD，是有某个权威机构制定，供特定行业或者公众使用。公用DTD 通过PUBLIC关键字引入，而不是使用SYSTEM。另外还要在增加一个标识名。语法如下

**<!DOCTYPE 根元素 PUBLIC "DTD标识名" "公用DTD的URI"*>***

#### 【示例6】MyBatis开发中引入公用DTD

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"*>*** <**configuration**>  <**settings**>  <**setting name="logImpl" value="LOG4J"**/>  </**settings**>  <**typeAliases**>  <**package name="com.bjsxt.user.pojo"**></**package**>  </**typeAliases**>  </**configuration**> |

在DTD中定义语义约束简单易用，但是也具有一些明显的劣势：DTD可以定义XML文档的结构，却无法对XML元素内容进行约束，比如，如果希望某个XML元素的内容是日期类型，希望内容必须是正整数，希望某个子元素最多出现3次，就无能为力了。这就需要使用XML Schema来进行语义约束了。

### 2.2 XML Schema约束

DTD和XML Schema是两种XML定义语义约束的工具，二者各有特色：DTD简单易用，但是功能相对较弱。XML Schema采用XML文档来定义语义约束，要复杂一些，但是功能强大的多。XML Schema指定丰富的类型，而且允许开发者自定义数据类型，因此完全可以处理更加复杂的语义约束场景。

XML Schema简称XSD(XML Schema Definition )，是DTD的替代者，既可以定义XML文档结构，也可以定义XML文档的内容约束。优势体现在：

* 可读性强：本身就是一个XML文档
* 支持数据类型：比如日期类型，并且限制日期范围都没有问题
* 可扩展：导入其他的Schema，自定义数据类型、一个XML文档使用多个XML Schema

JavaEE、JavaEE开源框架中都大量使用了XML文档，其语义约束也己经陆续从之前的DTD约束升级为Schema约束。

#### 【示例7】引入XML Schema的XML文件 shiporder.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="ISO-8859-1"***?>* <**shiporder orderid="889923"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="shiporder.xsd"**>  <**orderperson**>George Bush</**orderperson**>  <**shipto**>  <**name**>John Adams</**name**>  <**address**>Oxford Street</**address**>  <**city**>London</**city**>  <**country**>UK</**country**>  </**shipto**>  <**item**>  <**title**>Empire Burlesque</**title**>  <**note**>Special Edition</**note**>  <**quantity**>1</**quantity**>  <**price**>10.90</**price**>  </**item**>  <**item**>  <**title**>Hide your heart</**title**>  <**quantity**>1</**quantity**>  <**price**>9.90</**price**>  </**item**> </**shiporder**> |

这是存储一张发货单shiporder 的XML文件，数据包括一个发单人orderperson，字符串类型；一个收单人信息：shipto，复杂数据类型；多个发货单明细信息：item，也是复杂数据类型。

#### 【示例8】定义XML Schema文件shiporder.xsd

|  |
| --- |
| *<?***xml version="1.0" encoding="ISO-8859-1"** *?>* <**xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"**>  <!-- 简易元素的定义 -->  <**xs:element name="orderperson" type="xs:string"**/>  <**xs:element name="name" type="xs:string"**/>  <**xs:element name="address" type="xs:string"**/>  <**xs:element name="city" type="xs:string"**/>  <**xs:element name="country" type="xs:string"**/>  <**xs:element name="title" type="xs:string"**/>  <**xs:element name="note" type="xs:string"**/>  <**xs:element name="quantity" type="xs:positiveInteger"**/>  <**xs:element name="price" type="xs:decimal"**/>  <!-- 属性的定义 -->  <**xs:attribute name="orderid" type="xs:string"**/>  <!-- 复合元素的定义 -->  <**xs:element name="shipto"**>  <**xs:complexType**>  <**xs:sequence**>  <**xs:element ref="name"**/>  <**xs:element ref="address"**/>  <**xs:element ref="city"**/>  <**xs:element ref="country"**/>  </**xs:sequence**>  </**xs:complexType**>  </**xs:element**>  <**xs:element name="item"**>  <**xs:complexType**>  <**xs:sequence**>  <**xs:element ref="title"**/>  <**xs:element ref="note" minOccurs="0"**/>  <**xs:element ref="quantity"**/>  <**xs:element ref="price"**/>  </**xs:sequence**>  </**xs:complexType**>  </**xs:element**>  <**xs:element name="shiporder"**>  <**xs:complexType**>  <**xs:sequence**>  <**xs:element ref="orderperson"**/>  <**xs:element ref="shipto"**/>  <**xs:element ref="item" maxOccurs="unbounded"**/>  </**xs:sequence**>  <**xs:attribute ref="orderid" use="required"**/>  </**xs:complexType**>  </**xs:element**> </**xs:schema**> |

### 本节作业

1. DTD和XML Schema的作用和区别
2. 给存储如下数据的XML文件定义DTD文件

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | 名称 | 单价 | 颜色 | 尺寸 | 库存 |
| P001 | 蜘蛛王皮鞋 | 268 | 黑色 | 42 | 500 |
| P002 | ThinkPad x240 | 5678 | 黑色 | 12 | 50 |
| P003 | WD移动硬盘 | 568 | 蓝色 | 5 | 1000 |

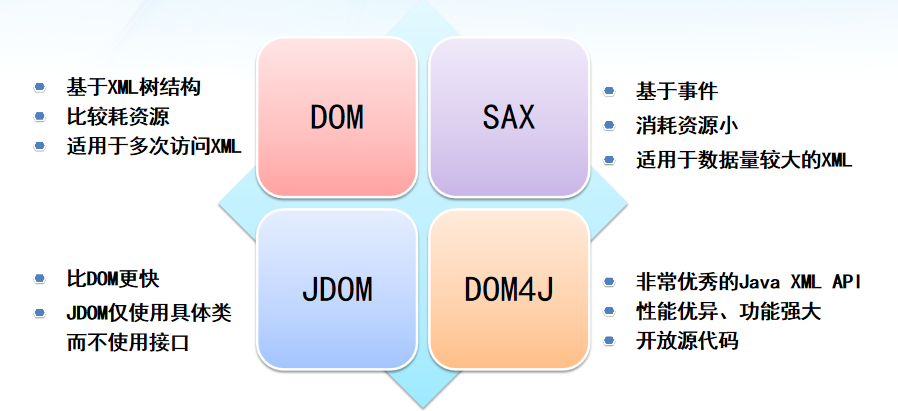
## 三、DOM解析XML

### 3.1 XML解析四种方式

XML作为一种数据传输工具，肯定离不开XML文档的读写。XML本身是结构化文档，如果依旧使用普通的IO流读写，效率低下，编程繁琐。目前常用的XML解析技术主要有四种。

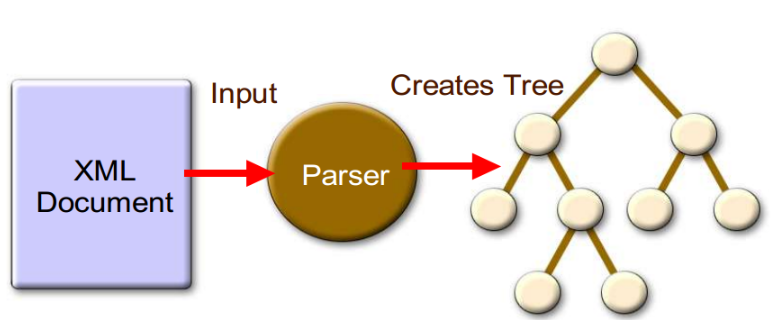
DOM和SAX是XML解析的两种规范，目前主流的XML解析器都会为DOM和SAX提供实现。

使用这两种技术解析XML比较繁琐，代码冗长，可读性也不高。所以Java领域中又出现了两个开源的XML解析器：DOM4J和JDOM，两个其实很有渊源，所以也非常相似。其中DOM4J是面向接口编程，而JDOM是面向实现编程。DOM4j比JDOM更灵活，性能表现也比较优异。



1. **DOM：Document Object Model 文档对象模型**

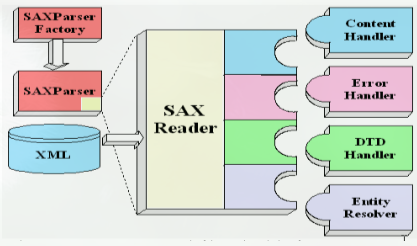
使用该技术解析XML文档时，会根据要操作的文档，构建一棵驻留在内存中的树，然后就可以使用DOM接口来操作这棵树。由于树是驻留在内存中，所以非常方便各种操作。但是也因为这棵树包含了XML文档的所有内容，是比较耗费资源的。该方式适合小文档的解析、适合多次访问的文档的解析。



1. **SAX：Simple API for XML**

是基于事件的解析，它是为了解决DOM解析的资源耗费而出现的。SAX在解析一份XML文档时，会依次出发文档开始、元素开始、元素结束、文档结束等事件，应用程序通过监听解析过程中所触发的事件即可获取XML文档的内容。该方式不需要事先调入整个文档，优势是占用资源少，内存消耗小，一般在解析数据量较大的文档是采用该方式。





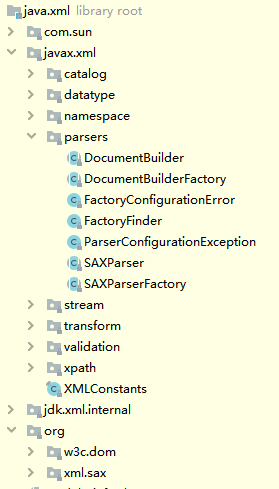
1. **DOM4J： DOM for Java**

开源的XML解析工具，完全支持DOM、SAX机制，具有性能优异、功能强大和操作简单等特点。越来越多的Java软件都在使用DOM4J处理XML文档。

1. **JDOM： Java DOM**

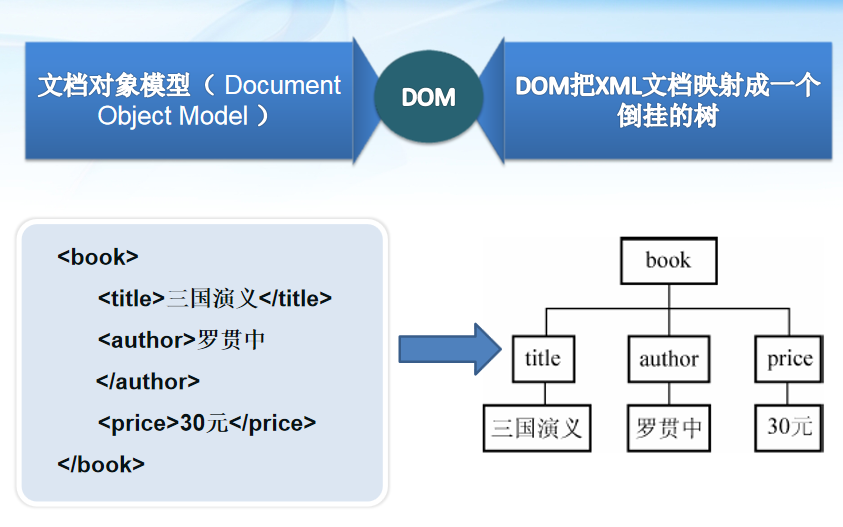
JDOM的目的是成为Java特定文档模型。行至半路，一部分人产生了新的想法，而这些想法又无法在JDOM中实现，干脆就从该项目中分离出来，单独去开发另外一套专属的XML API，这就是DOM4J。因此，两者具有相同的设计目的，用法也非常相似。从组中解决来看，JDOM的主要API以类为主，DOM4J的API以接口为主。

Java对DOM和SAX两种规范都提供了支持。Java解析XML文档的API称为JAXP（Java API for XMLProcessing），作为JDK的一部分发布。其中javax.xml.parsers包中提供了四个与DOM和SAX解析相关的类。如果使用DOM解析，就使用org.w3c.dom包下的类和接口。如果使用SAX解析，就使用org.xml.sax包下的类和接口。



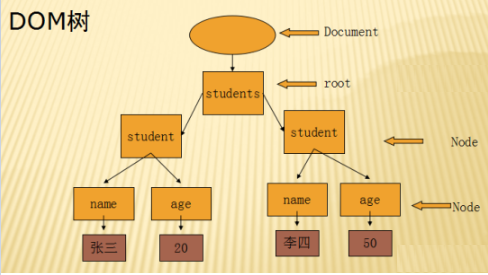
### 3.2 认识DOM

DOM，Document Object Model。将XML文档解析为一棵树，XML文档的节点对应DOM树的节点，节点之间保持父子、兄弟关系。并且DOM树中的每个节点都是一个对象，如果要解析该文档，使用面向对象的思想调用节点的属性和方法即可。可以对该树进行添加、查询、修改、删除等操作，最终会转换为对对应XML文档的操作。



Node类是DOM树中节点的父类。根据具体类型可以分为多个子类，比如元素节点类Element、属性节点类Attr、文本节点类Text等。另外还有注释类Comment、文档类Document。

Document类代表整个XML文档本身，对整个文档进行操作的入口。Document对象中包含一个根节点。



### 3.3 使用DOM解析XML

使用DOM可以解析XML，也可以对XML进行添加、修改、删除操作。因为DOM操作比较繁琐，代码冗长，这里我们只进行解析操作，理解XML解析的主要步骤，发现DOM解析的弊端。在后面DOM4J部分完成对XML文档的所有操作。

#### 【示例9】使用DOM解析XML

|  |
| --- |
| **public class** TestDOM {  **public static void** main(String[] args) **throws** ParserConfigurationException,  SAXException, IOException {  // 1.创建DOM解析器工厂  DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance*();  // 2.由DOM解析器工厂创建DOM解析器  DocumentBuilder db = dbf.newDocumentBuilder();  // 3.由DOM解析器解析文档，生成DOM树  Document doc = db.parse(**"module2/student2.xml"**);  // 4.解析DOM树，获取文档内容（元素 属性 文本）  // 4.1获取根元素students  NodeList studentsList = doc.getElementsByTagName(**"students"**);  // NodeList studentsList = doc.getChildNodes();  // System.out.println(studentsList.getLength());  Node studentsNode = studentsList.item(0);// students根节点  // 4.2获取students中所有的子元素student  NodeList studentList = studentsNode.getChildNodes();  // System.out.println(studentList.getLength());//空白也算节点  // 4.3对每个student进行处理  **for** (**int** i = 0; i < studentList.getLength(); i++) {  // 获取第i个节点  Node studentNode = studentList.item(i);  // System.out.println(studentNode.getNodeType());  **if** (studentNode.getNodeType() == Node.***ELEMENT\_NODE***) {// Node.ELEMENT\_NODE  // :1  // 获取每个学生的id属性及其值  Element stuElem = (Element) studentNode;  String value = stuElem.getAttribute(**"id"**);  System.***out***.println(**"id----->"** + value);  // 获取每个学生的name，age，score  NodeList nasList = stuElem.getChildNodes();  // System.out.println(nasList.getLength());  **for** (**int** j = 0; j < nasList.getLength(); j++) { // 7  // 获取第j个节点  Node nasNode = nasList.item(j);  // 只处理Element，不处理空白  **if** (nasNode.getNodeType() == Node.***ELEMENT\_NODE***) {// 3  // 得到元素节点  Element nasElem = (Element) nasNode;  // 获取元素名称  String name = nasElem.getNodeName();  // 获取元素的文本  String content = nasElem.getTextContent();  // 输出元素的名称和文本  System.***out***.println(name + **"----"** + content);  }  }  System.***out***.println();  }  }  } } |

**缺点**

1. 繁琐：前面三个步骤相同，也需要书写一遍。能否封装好
2. 繁琐：getChildNodes()不仅包括Element，也包括空白形成的Text，遍历时需要进行筛选
3. 繁琐：getChildNodes()也包括注释、也包括外部DTD引用,大部分情况下并不是用户需要的

**解决：DOM4J**

### 本节作业

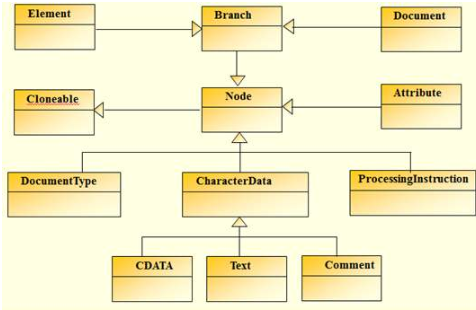
1. XML四种解析方式及其特点
2. 如何理解DOM树
3. DOM解析的缺点

## 四、使用DOM4J解析XML

### 4.1 认识DOM4J

DOM4J是一套开源的XML解析工具。与利用DOM、SAX、JAXP机制来解析XML相比，DOM4J 表现更优秀，具有性能优异、功能强大和极端易用使用的特点，只要懂得DOM基本概念，就可以通过DOM4J的API文档来解析XML。DOM4J是一套开源的API。实际项目中，往往选择DOM4J来作为解析XML的利器。

DOM4J在很大程度上简化了XML的处理方式。从表面上看，DOM4J有点类似DOM的解析机制，也将XML文档转换为一棵结构化树（称为DOM4J树吧），但是DOM4J的处理方式比DOM树更简单。



|  |  |
| --- | --- |
| Attribute | 定义了 XML 的属性。 |
| Node | DOM4J树中所有节点的根接口 |
| Branch | 指包含子节点的节点。如元素(Element)和文档(Docuemnts) |
| Document | 代表XML 文档 |
| Element | 代表XML 元素 |
| CharacterData | 所有文本元素的父接口。如CDATA，Comment, Text. |
| CDATA | 代表 XML CDATA 区域 |
| Comment | 代表 XML 注释内容 |
| DocumentType | 代表 XML DOCTYPE 声明 |
| ProcessingInstruction | 代表 XML 处理指令 |
| Attribute | 代表XML元素的属性 |

### 4.2 使用DOM4J解析XML

#### 【示例10】使用DOM4J解析XML

|  |
| --- |
| **public static void** main(String[] args) **throws** DocumentException {   *//1.根据xml文件创建DOM树* SAXReader reader = **new** SAXReader();  *//2、解析指定的xml文件* Document dom = reader.read(**new** File(**"D:\\ASXTCourse\\JavaSE2\\xml\\src\\student.xml"**));  *//3、获得根节点元素对象 就是students* Element rootElement = dom.getRootElement();  *//4、获得students下的子节点  // List<Element> elements = rootElement.elements();* List<Element> elements2 = rootElement.elements(**"student"**);  **for** (Element e:elements2) {  *// System.out.println(e.attribute("id").getValue());//获得指定节点属性的值  //student元素下的子节点* List<Element> list2 = e.elements();  **for** (Element e2:list2) {  System.***out***.println(e2.getName());*//获得节点元素名称* System.***out***.println(e2.getText());*//获得节点的文本内容* System.***out***.println(**"-----------------------"**);  }  } } |

技能点1：DOM4J对底层原始的XML解析器进行了高度封装，正是这种封装简化了XML处理。在DOM4J的org.dom4j.io包下提供了如下几个类：

* DOMReader：根据W3C的DOM树创建DOM4J树
* SAXReader：基于SAX解析机制解析一份XML文档，并将其转换为DOM4J树

技能点2：获取属性

* 获取所有属性 List<Attribute> attributes = elem.attributes();
* 获取指定属性 Attribute attr = elem.attribute("id");
* attr.getName()+":"+attr.getValue() 获取属性名和属性值

技能点3：获取元素

* Element rootElem = doc.getRootElement(); 获取根元素
* List<Element> stuList = rootElem.elements(); 获取所有名称的子元素列表
* List<Element> stuList = rootElem.elements("student");获取指定名称子元素列表
* String ename = subElem.getName(); 获取元素名称
* String etext = subElem.getText(); 获取元素文本

### 4.3 使用DOM4J完成添加操作

#### 【示例11】创建一个新的XML文件

|  |
| --- |
| **public class** TestDom4j2 {  **public static void** main(String[] args) **throws** DocumentException, IOException {  //1.创建一个DocumentFactory对象  //DocumentFactory factory = new DocumentFactory();  //2.创建一个Document对象  //Document doc = factory.createDocument();  Document doc = DocumentHelper.*createDocument*();  doc.addComment(**"student list"**);  //3.获取DOM树的根节点students  Element rootElem = doc.addElement(**"students"**);  //4.在DOM树中给students添加子节点  Element stuElem = rootElem.addElement(**"student"**);  stuElem.addAttribute(**"id"**,**"003"**);//id属性  Element stuAgeElem = stuElem.addElement(**"age"**);//age子元素  stuAgeElem.setText(**"30"**);  Element stuNameElem = stuElem.addElement(**"name"**); //name子元素  stuNameElem.setText(**"张三"**);  Element stuScoreElem = stuElem.addElement(**"score"**);//score子元素  stuScoreElem.setText(**"97"**);  //5.将DOM树最新数据写入XML文件  OutputFormat format = OutputFormat.*createPrettyPrint*();//精致美观格式  //OutputFormat format = OutputFormat.createCompactFormat();//紧密压缩格式  format.setEncoding(**"utf-8"**);  Writer fw = **new** FileWriter(**"module2/student3.xml"**);  //XMLWriter xmlWriter = new XMLWriter(System.out,format);  XMLWriter xmlWriter = **new** XMLWriter(fw,format);  xmlWriter.write(doc);  xmlWriter.close();  } } |

技能点1：如何创建新文档Document：

* DocumentFactory：使用了工厂模式
* DocumentHelper：底层还是调用了DocumentFactory

技能点2：如何添加子元素

* Element stuElem = rootElem.addElement("student");
* stuElem.addAttribute("id","003");//id属性
* Element stuAgeElem = stuElem.addElement("age");//age子元素
* stuAgeElem.setText("30");

技能点3：如何写数据到XML文件

* XMLWriter xmlWriter = new XMLWriter(fw,format);
* createPrettyPrint：精致美观格式，带缩进、有换行，格式美观
* createCompactFormat：紧密压缩格式，无缩进、无换行，不推荐

#### 【示例12】添加元素到已存在的文件（默认最后一个子元素）

|  |
| --- |
| **public class** TestDom4j3 {  **public static void** main(String[] args) **throws** DocumentException, IOException {  //1.根据xml文件创建DOM树  SAXReader reader = **new** SAXReader();  File file = **new** File(**"module2/student3.xml"**);  Document doc = reader.read(file);  //2.获取DOM树的根节点  Element rootElem = doc.getRootElement();  //3.在DOM树中给students添加子节点  Element stuElem = rootElem.addElement(**"student"**);  stuElem.addAttribute(**"id"**,**"005"**);  Element stuAgeElem = stuElem.addElement(**"age"**);  stuAgeElem.setText(**"32"**);  Element stuNameElem = stuElem.addElement(**"name"**);  stuNameElem.setText(**"王五"**);  Element stuScoreElem = stuElem.addElement(**"score"**);  stuScoreElem.setText(**"100"**);  //4.将DOM树最新数据写入XML文件  OutputFormat format = OutputFormat.*createPrettyPrint*();  format.setEncoding(**"utf-8"**);  Writer fw = **new** FileWriter(**"module2/student3.xml"**);  XMLWriter xmlWriter = **new** XMLWriter(fw,format);  xmlWriter.write(doc);  xmlWriter.close();  } } |

#### 【示例13】添加元素到已存在的文件（指定位置）

|  |
| --- |
| **public class** TestDom4j4 {  **public static void** main(String[] args) **throws** DocumentException, IOException {  //1.根据xml文件创建DOM树  SAXReader reader = **new** SAXReader();  File file = **new** File(**"module2/student2.xml"**);  Document doc = reader.read(file);  //2.获取DOM树的根节点students  Element rootElem = doc.getRootElement();  //3.在DOM树中给students添加子节点到指定位置  List<Element> list = rootElem.elements(**"student"**);  Element stuElem =DocumentHelper.*createElement*(**"student"**);  stuElem.addAttribute(**"id"**,**"002"**);  Element stuAgeElem = stuElem.addElement(**"age"**);  stuAgeElem.setText(**"30"**);  Element stuNameElem = stuElem.addElement(**"name"**);  stuNameElem.setText(**"李四"**);  Element stuScoreElem = stuElem.addElement(**"score"**);  stuScoreElem.setText(**"90"**);  list.add(1,stuElem);  //4.将DOM树最新数据写入XML文件  OutputFormat format = OutputFormat.*createPrettyPrint*();  format.setEncoding(**"utf-8"**);  Writer fw = **new** FileWriter(**"module2/student3.xml"**);  XMLWriter xmlWriter = **new** XMLWriter(fw,format);  xmlWriter.write(doc);  xmlWriter.close();  } } |

技能点1：添加元素到指定位置

List<Element> list = rootElem.elements(**"student"**);  
 Element stuElem =DocumentHelper.*createElement*(**"student"**);  
 list.add(1,stuElem);

### 4.4 使用DOM4J完成删除修改操作

#### 【示例14】使用DOM4J删除指定元素

|  |
| --- |
| **public class** TestDom4j5 {  **public static void** main(String[] args) **throws** DocumentException, IOException {  //1.根据xml文件创建DOM树  SAXReader reader = **new** SAXReader();  File file = **new** File(**"module2/student3.xml"**);  Document doc = reader.read(file);  //2.获取DOM树的根节点students  Element rootElem = doc.getRootElement();  //3.在DOM树中给students删除子节点  List<Element> list = rootElem.elements(**"student"**);  **for** (**int** i = 0; i <list.size() ; i++) {  Element elem = list.get(i);  **if**(elem.attribute(**"id"**).getValue().equals(**"003"**)){  //list.remove(elem);  rootElem.remove(elem);  **break**;  }  }  //4.将DOM树最新数据写入XML文件  OutputFormat format = OutputFormat.*createPrettyPrint*();  format.setEncoding(**"utf-8"**);  Writer fw = **new** FileWriter(**"module2/student3.xml"**);  XMLWriter xmlWriter = **new** XMLWriter(fw,format);  xmlWriter.write(doc);  xmlWriter.close();  } } |

技能点1：删除指定元素方法1：采用集合的删除操作

List<Element> list = rootElem.elements(**"student"**);  
 list.remove(elem);

技能点2：删除指定元素方法2：父节点删除子节点

Element rootElem = doc.getRootElement();  
 rootElem.remove(elem);

#### 【示例15】使用DOM4J修改指定元素

|  |
| --- |
| **public class** TestDom4j7 {  **public static void** main(String[] args) **throws** DocumentException, IOException {  //1.根据xml文件创建DOM树  SAXReader reader = **new** SAXReader();  File file = **new** File(**"module2/student3.xml"**);  Document doc = reader.read(file);  //2.获取DOM树的根节点students  Element rootElem = doc.getRootElement();  //3.在DOM树中给修改指定student节点  List<Element> list = rootElem.elements(**"student"**);  **for** (**int** i = 0; i <list.size() ; i++) {  Element elem = list.get(i);  **if**(elem.attribute(**"id"**).getValue().equals(**"005"**)){  elem.attribute(**"id"**).setValue(**"006"**);  Element nameElem = elem.element(**"name"**);  nameElem.setText(**"赵六"**);  **break**;  }  }  //4.将DOM树最新数据写入XML文件  OutputFormat format = OutputFormat.*createPrettyPrint*();  format.setEncoding(**"utf-8"**);  Writer fw = **new** FileWriter(**"module2/student3.xml"**);  XMLWriter xmlWriter = **new** XMLWriter(fw,format);  xmlWriter.write(doc);  xmlWriter.close();  } } |

技能点1：修改指定元素

elem.attribute(**"id"**).setValue(**"006"**); 修改元素的属性值  
 Element nameElem = elem.element(**"name"**); 修改元素的子元素的文本  
 nameElem.setText(**"赵六"**);

### 本节作业

1. 使用DOM4J解析XML
2. 使用DOM4J添加子元素
3. 使用DOM4J删除子元素
4. 使用DOM4J修改子元素