案例：<https://gitee.com/zx19890628/spring-boot-example/tree/master/lab_066_util_xml>

XML全称为Extensible Markup Language，意思是可扩展的标记语言。XML语法上和HTML比较相似，但HTML中的元素是固定的，而XML的标签是可以由用户自定义的。

W3C在1998年2月发布1.0版本，2004年2月又发布1.1版本，但因为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

# XML基础

## XML是什么？作用是什么？

1、XML是指可扩展标记语言（Extensible Markup Language），用户自定义的标签.相对于HTML来讲的。

2、XML被设计的宗旨是表示数据。HTML是用来显示数据的。目前经常使用的XML版本是1.0

3、XML除了表示数据外。在实际的企业开发中，主要用XML作为程序的配置文件。

## XML的基本语法

### 文档声明

作用：用于标识该文档是一个XML文档。

注意事项：声明必须出现在文档的第一行（之前连空行都不能有，也不能有任何的注释）

文档声明必须为<?xml开头，以?>结束；

文档声明必须从文档的0行0列位置开始；

文档声明只有三个属性：

versioin：指定XML文档版本。必须属性，因为我们不会选择1.1，只会选择1.0；

encoding：指定当前文档的编码。可选属性，默认值是utf-8；

standalone：指定文档独立性。可选属性，默认值为yes，表示当前文档是独立文档。如果为no表示当前文档不是独立的文档，会依赖外部文件。

### 元素

结束标签不能省略

一个XML文档必须且只能有一个根标签

XML文档中不会忽略回车和换行及空格

标签的命名规范：元素(标签)的名称可以包含字母、数字、减号、下划线和英文句点。严格区分大小写。

### 元素的属性

元素的属性取值一定要用引号引起来（单引号或双引号）

### 注释

与HTML中的注释完全一致：<!--这是注释-->

注释不能嵌套

### CDATA区

CDATA是Character Data的缩写。

作用：CDATA区中的东东都是文本。

语法：

<![CDATA[

文本内容

]]>

### 特殊字符

& &amp;

< &lt; (less than)

> &gt: (great than)

" &quot;

' &apos;

### 处理指令(PI:Processing Instruction)

处理指令，简称PI(Processing Instruction)。

作用：用来指挥软件如何解析XML文档。

语法：必须以“<?”作为开头，以“?>”作为结尾。

## XML的约束

1、格式良好的XML文档：符合XML语法的。

2、有效的XML文档：遵循约束规范的。

格式良好的不一定是有效的，但有效的必定格式良好。

## DTD的基本语法（看懂即可）

1、DTD：Document Type Definition

2、作用：约束XML的书写规范。

3、DTD文件保存到磁盘时，必须使用UTF-8编码

4、如何引入外部的DTD文档来约束当前的XML文档

DTD文件在本地：<!DOCTYPE 根元素名称 SYSTEM "DTD文件的路径">

DTD文件在网络上：<!DOCTYPE 根元素名称 PUBLIC "DTD名称" "DTD的路径URL">

## 格式良好的XML文档

格式良好的XML就是格式正确的XML文档，只有XML的格式是良好的，XML解释器才能解释它。下面是对格式良好XML文档的要求：

必须要有XML文档声明；

必须且仅能有一个根元素；

元素和属性的命名必须遵循XML要求：

XML命名区分大小写，例如<a>和<A>是两个不同的元素；

名称中可以包含：字母、数字、下划线、减号，但不能以数字、减号开头；

不能以xml开头，无论是大写还是小写都不可以，例如<xml>、<Xml>、<XML>都是错误的；

不能包含空格，例如<ab cd>是错误的。

元素之间必须合理包含，例如：<a><b>xxx</b></a>是合理的，而<a><b>xxx</a></b>就是错误的包含。

## DTD的语法细节

### 定义元素

语法：<!ELEMENT 元素名称 使用规则>

使用规则：

(1).(#PCDATA):指示元素的主体内容只能是普通的文本.(Parsed Character Data)

(2).EMPTY:指示元素的不能有主体内容。

(3).ANY:用于指示元素的主体内容为任意类型

(4).(子元素)：指示元素中包含的子元素

如果子元素用逗号分开，说明必须按照声明顺序去编写XML文档

如果子元素用“|”分开，说明任选其一。

用+、\*、？来表示元素出现的次数依次：1次或多次、多次，1次到0次

没有+\*?的表示必须且只能出现一次。

### 定义元素的属性(attribute)

语法：<!ATTLIST 哪个元素的属性

属性名1 属性值类型 设置说明

属性名2 属性值类型 设置说明>

属性值类型：

CDATA：说明该属性的取值为一个普通文本

ENUMERATED (DTD没有此关键字)：

语法：<!ATTLIST 元素名称 (值1|值2) "值1">

ID:属性的取值不能重复

设置说明：

#REQUIRED：表示该属性必须出现

#IMPLIED：属性可有可无

#FIXED:表示属性的取值为一个固定值 语法：#FIXED "固定值"

直接值：表示属性的取值为该默认值

### 定义实体

关键字ENTITY

实体的定义分为引用实体和参数实体

引用实体：

作用：在DTD中定义，在XML中使用

语法：<!ENTITY 实体名称 "实体内容">

在XML中使用：&实体名称;

参数实体：

作用：在DTD中定义，在DTD中使用

语法：<!ENTITY % 实体名称 "实体内容">

在DTD中使用：%实体名称;

# Xml解析

解析器的概念：解析器就是读取xml的一系列类.

xml==> 记录数据,配置文件.

xml本质还是一个文本.我们在使用它不管干什么.都涉及到读取的问题.让我们来真正做读取.

可以做,但是太麻烦. 而读取这个过程就叫做解析

对数据常见操作: CRUD create read update delete

从这个角度看.解析xml指的就是读取xml数据.

## XML解析方式概述

xml常用的四种解析技术：DOM、SAX、JDOM、DOM4J

### DOM（Document Object Model）

文档对象模型分析方式。以层次结构（类似于树型）来组织节点和信息片段，映射XML文档的结构，允许获取和操作文档的任意部分。

是W3C的官方标准。非常适合XML的CRUD操作，适用于操作较小文件大小的XML文件

优点：

1、允许应用程序对数据和结构做出更改。

2、访问是双向的，可以在任何时候在树中上下导航，获取和操作任意部分的数据。

缺点：

1、通常需要加载整个XML文档来构造层次结构，消耗资源大

### SAX（Simple API for XML）非常不好用，用着蛋疼

流模型中的推模型分析方式。通过事件驱动，每发现一个节点就引发一个事件，通过回调方法完成解析工作，解析XML文档的逻辑需要应用程序完成。

适用于查询XML文件，对文件的大小没有过多的限制。

优点：

1、不需要等待所有数据都被处理，分析就能立即开始。

2、只在读取数据时检查数据，不需要保存在内存中。

3、可以在某个条件得到满足时停止解析，不必解析整个文档。

4、效率和性能较高，能解析大于系统内存的文档。

缺点：

1、需要应用程序自己负责TAG的处理逻辑（例如维护父/子关系等），使用麻烦。

2、单向导航，很难同时访问同一文档的不同部分数据，不支持XPath。

### JDOM（Java-based Document Object Model）

Java特定的文档对象模型。自身不包含解析器，使用SAX。

优点：

1、使用具体类而不是接口，简化了DOM的API。

2、大量使用了Java集合类，方便了Java开发人员。

缺点：

1、没有较好的灵活性。

2、性能较差。

### DOM4J（Document Object Model for Java）

简单易用，采用Java集合框架，并完全支持DOM、SAX和JAXP。

优点：

1、大量使用了Java集合类，方便Java开发人员，同时提供一些提高性能的替代方法。

2、支持XPath。

3、有很好的性能。

缺点：

1、大量使用了接口，API较为复杂。

2、常用解析开发包：

## JAXP

是JDK提供的解析XML的相关接口和类。

JAXP(Java API for XMLProcessing，意为XML处理的Java API)是Java XML程序设计的应用程序接口之一，它提供解析和验证XML文档的能力。

JAXP是在Java社区进程下开发的，包括JSR 5 (JAXP 1.0)和 JSR 63 (JAXP 1.1和1.2)两个规范。

JAXP解析XML的四种基本接口为:

### org.w3c.dom:提供DOM方式解析XML的标准接口

DOM接口可能最容易理解。解析器解析整个XML文档并构造出该文档的一个完整的内存表示，使用类对文档对象模型(DOM)级别2核心规范中的概念进行建模。

DOM解析器被称作DocumentBuilder，因为它构建一个内存中的Document表示。javax.xml.parsers.DocumentBuilder是由

javax.xml.parsers.DocumentBuilderFactory创建的。DocumentBuilder创建org.w3c.dom.Document的实例，Document是一个包含XML文档中的

所有节点的树。结构中的每一个树节点实现org.w3c.dom.Node接口。树节点有许多不同类型的，表示XML文档中不同的数据类型。

Node的类型，包括：

ELEMENT\_NODE 元素节点

ATTRIBUTE\_NODE 属性节点

TEXT\_NODE 文本节点

CDATA\_SECTION\_NODE

ENTITY\_REFERENCE\_NODE

ENTITY\_NODE 空节点

PROCESSING\_INSTRUCTION\_NODE

COMMENT\_NODE

DOCUMENT\_NODE

DOCUMENT\_TYPE\_NODE

DOCUMENT\_FRAGMENT\_NODE

NOTATION\_NODE

### org.xml.sax:提供SAX方式解析XML的标准接口

SAX解析器被称作SAXParser，SAXParser是由javax.xml.parsers.SAXParserFactory创建的。与DOM解析器不同，SAX解析器并不创建XML文档的

内存表示，因此要更快使用更少的内存。而是，SAX解析器通过调用回调方法将XML文档结构告知客户端，也就是说，通过调用提供给解析器的

org.xml.sax.helpers.DefaultHandler实例上的方法。

DefaultHandler类实现了ContentHandler，ErrorHandler，DTDHandler以及EntityResolver等接口。

大多数客户端实现ContentHandler接口中定义的方法，当SAX解析器在XML文档中遇到元素时，相应的方法将被调用。接口中最重要的方法有:

startDocument()和endDocument()方法，当XML文档的开始和结束时被调用。

startElement()和endElement()方法，当一个文档元素开始和结束时被调用。

characters()方法，调用处理XML文档元素的开始标签和结束标签之间的文本数据。

客户端提供一个DefaultHandler的子类，实现这些方法，并在方法中对数据进行处理，可能会将数据存入数据库，或写出到流中。

在解析过程中，解析器可能需要访问外部文档。因此可能会使用XML Catalog将经常使用的XML文档存储在本地的缓存中。

### XML流API或StAX接口(是JDK 6的一部分，为JDK 5提供单独的包)

### 除了解析接口，JAXP还提供了XSLT接口用来对XML文档进行数据和结构的转换。

XSLT(可扩展样式表转换语言)允许将XML文档转换为数据其他形式。应用程序使用javax.xml.transform包中的接口可以进行XSLT转换。

接口的主要特性包括:

a.工厂类javax.xml.transform.TransformerFactory可以使应用动态地选择使用哪一个XSLT处理器。

b.TransformerFactory上的方法用来创建javax.xml.transform.Templates对象, 表示样式表的编译后的形式。这是一个线程安全的对象，

可以重复使用，顺序或并发，在多个源文档上应用同一个样式表(或用用一个源文档，不同的参数)

c.Templates上的方法可以创建javax.xml.transform.Transformer，表示样式表的可执行形式。Transformer不可以在线程间共享，

虽然也是可重用的。Transformer提供方法设置样式表参数和序列化选项(例如，输出是否缩进)，以及一个实际运行转换的方法。

JAXP定义了javax.xml.transform.Source和javax.xml.transform.Result两个抽象接口来表示转换的输入和输出。某种程度上，

这是非常规使用Java接口，这是因为并不期待一个处理器会接受任何实现该接口的类，每一个处理器可以选择支持那些Source和Result的处理。

实际上所有JAXP处理器支持三种标准类型的Source (DOMSource，SAXSource，StreamSource)以及三种标准类型的Result

(DOMResult，SAXResult，StreamResult)以及处理器自己的实现。

## JAXP进行DOM解析的案例

1.基本工具类

//1.获得DOM创建器工厂

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

//2.创建一个DOM创建器

DocumentBuilder builder = factory.newDocumentBuilder();

//3.解析XML获得DOM树

Document doc = builder.pares("src/book.xml");

2.查询

//1.按照节点名称(HTML中的标签的名称)获得所有节点的集合

NodeList authors = doc.getElementsByTagName("作者");

//2.获得集合中的第一个

//Node author = authors.item(0)

//3.获得这个节点的文本

String authorName = author.getTextContent();

//4.将节点强制转换为元素\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*推荐将节点强制转换为元素，很多操作很方便

Element ele = (Element) author;

//5.获得属性值

String attribute = ele.getAttribute("年龄");

3.修改 ，完成后需要保存到本地文件中

//1.修改节点的文本值

author.setTextContent("张三");

//2修改属性,该出行存在则修改，不存在则创建

Element e = (Element)book;

e.setAttribute("出版社", "不告诉你出版社");

4.创建

//1.创建一个节点

Element e = doc.createElement("批发价");

e.setTextContent("58.00元");

//2.插入到某个节点的前面

price.getParentNode().insertBefore(e, price);

//3.增加一个元素

book.appendChild(e)

5.删除

//删除节点price。

price.getParentNode().removeChild(price)

6.保存修改过的XML

Transformer transformer = TransformerFactory.newInstance().newTransformer();

//doc是修改过的XML ，StreamResult是保存文件的路径

transformer.transform(new DOMSource(doc), new StreamResult("src/book.xml"));

## JAXP进行SAX解析

事件驱动.在读取的过程中.将遇到的文档的各个部分的内容.拆分各个事件.

当我们遇到什么事件的时候,可以选择做相应事.

xml:

文档开始事件 startDocument

文档结束事件 endDocument

元素开始事件 startElement

元素结束事件 endElement

文本结束事件 characters

1.在开发的时候必须继承DefaultHandler或其子类

public class StudentDao extends DefaultHandler {

@Override

public void startDocument() throws SAXException {

super.startDocument();

System.out.println("文档开始解析!");

}

@Override

public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

}

@Override

public void characters(char[] ch, int start, int length) throws SAXException {

}

@Override

public void endElement(String uri, String localName, String qName) throws SAXException {

}

@Override

public void endDocument() throws SAXException {

super.endDocument();

System.out.println("文档结束解析!");

}

public static void main(String[] args) {

StudentDao dao = new StudentDao();

SAXParser saxParser = SAXParserFactory.newInstance().newSAXParser();

saxParser.parse("src/student.xml", dao);

List<Student> queryAll = dao.queryAll();

}

}

## DOM和SAX 对比

1.DOM 内存占用较高. SAX在解析过程中,遇到事件会触发一段我们定义的代码.

代码运行完,那么解析过程也完成.内存占用非常小.

2.因为DOM 会把所有对象放到内存中.我们可以对内存中的对象进行增删改查等等一系列操作.

而sax只能做查询.

## DOM4j解析

DOM4J是针对Java开发人员专门提供的XML文档解析规范，它不同与DOM，但与DOM相似。DOM4J针对Java开发人员而设计，所以对于Java开发人员来说，

使用DOM4J要比使用DOM更加方便。在DOM4J中，也有Node、Document、Element等接口，结构上与DOM中的接口比较相似。但它们是不同的类

详细看课件“XML解析.doc”

# XStream

## XStream的作用

XStream可以把JavaBean对象转换成XML！

通常服务器向客户端响应的数据都是来自数据库的一组对象，而我们不能直接把对象响应给响应端，所以我们需要把对象转换成XML再响应给客户端，这时就需要使用XStream组合了。

## XStream入门

为了演示XStream的作用，我们需要先写两个类，Province和City。

City.java

public class City {

private String name;

private String description;

……

}

Province.java

public class Province {

private String name;

private List<City> cities = new ArrayList<City>();

public void addCity(City city) {

cities.add(city);

}

……

}

接下来，我们需要写一个main()，创建一个List，List中存放两个Province对象！最终我们把List转换成XML。

Province p1 = new Province("辽宁省");

p1.addCity(new City("沈阳", "shenyang"));

p1.addCity(new City("大连", "dalian"));

Province p2 = new Province("吉林省");

p2.addCity(new City("长春", "changchen"));

p2.addCity(new City("白城", "baicheng"));

List<Province> list = new ArrayList<Province>();

list.add(p1);

list.add(p2);

### XStream相关JAR包

我们可以到http://xstream.codehaus.org/地址去下载XStream安装包！

XStream的必导JAR包：

?核心JAR包：xstream-1.4.7.jar；

?必须依赖包：xpp3\_min-1.1.4c（XML Pull Parser，一款速度很快的XML解析器）；

### 使用XStream把Java对象转换成XML

下面是使用XStream转换list为XML的代码：

XStream xstream = new XStream();

String s = xstream.toXML(list);

System.out.println(s);

<list[与对象名对应！]>

<cn.itcast.xstream.demo1.Province[与Province类名对应]>

<name[与Province类的name属性对应]>辽宁省</name>

<cities[与Province类的cities属性对应]>

<cn.itcast.xstream.demo1.City[与City类名对应]>

<name[与City类的name属性对应]>沈阳</name>

<description[与City类的description属性对应]>shenyang</description>

</cn.itcast.xstream.demo1.City>

<cn.itcast.xstream.demo1.City>

<name>大连</name>

<description>dalian</description>

</cn.itcast.xstream.demo1.City>

</cities>

</cn.itcast.xstream.demo1.Province>

<cn.itcast.xstream.demo1.Province>

<name>吉林省</name>

<cities>

<cn.itcast.xstream.demo1.City>

<name>长春</name>

<description>changchen</description>

</cn.itcast.xstream.demo1.City>

<cn.itcast.xstream.demo1.City>

<name>白城</name>

<description>baicheng</description>

</cn.itcast.xstream.demo1.City>

</cities>

</cn.itcast.xstream.demo1.Province>

</list>

　　也就是说，XStream是根据对象名、类名、属性名来生成XML文档的！

### alias用法

大家也看到了，生成的XML中，与类名对应的元素名称包含了包名部分，这很不好看！若想自定义生成的元素名称，需要使用XStream为类名提供别名：

xstream.alias("province", Province.class);[把Province类对应的元素名命名为province]

xstream.alias("china", List.class); [把list对象对应的元素名称命名为china]

xstream.alias("city", City.class);[把City类对象的元素名称命名为city]

<china>[原来是list，对应list对象名称，现在已经为china]

<province>[原来是类的全名]

<name>辽宁省</name>

<cities>

<city>[原来是类的全名]

<name>沈阳</name>

<description>shenyang</description>

</city>

<city>

<name>大连</name>

<description>dalian</description>

</city>

</cities>

</province>

<province>

<name>吉林省</name>

<cities>

<city>

<name>长春</name>

<description>changchen</description>

</city>

<city>

<name>白城</name>

<description>baicheng</description>

</city>

</cities>

</province>

</china>

### 把子元素变为元素属性

例如我们需要把<province>子元素<name>变成：<province name=””>样式，那么需要调用如下方法：

xstream.useAttributeFor(Province.class, "name");[让Province类的name成员以属性形式出现！]

<china>

<province name="辽宁省">[Province类的name成员以元素的属性形式出现。]

<cities>

<city>

<name>沈阳</name>

<description>shenyang</description>

</city>

<city>

<name>大连</name>

<description>dalian</description>

</city>

</cities>

</province>

<province name="吉林省">

<cities>

<city>

<name>长春</name>

<description>changchen</description>

</city>

<city>

<name>白城</name>

<description>baicheng</description>

</city>

</cities>

</province>

</china>

### 去除集合属性对应元素

大家可能已经发现了，因为Pronvice类有一个cities成员，所以生成了<cities>元素，但这个元素对XML文档而言没有什么意义，所以我们希望把它去除！

xstream.addImplicitCollection(Province.class, "cities");[隐藏Province类的名为cities的集合元素]

<china>

<province name="辽宁省">[province元素下再没有ciites元素，而是由province元素直接包含city元素]

<city>

<name>沈阳</name>

<description>shenyang</description>

</city>

<city>

<name>大连</name>

<description>dalian</description>

</city>

</province>

<province name="吉林省">

<city>

<name>长春</name>

<description>changchen</description>

</city>

<city>

<name>白城</name>

<description>baicheng</description>

</city>

</province>

</china>

### 让类的成员不生成对应XML元素

到现在为止，我们都是每个类，每个成员都有对应的元素（或属性）存在，但有时我们并不希望某些类的成员在对应的XML文档中出现，例如我们不希望City类的description成员出现在XML文档中，可以使用下面方法：

xstream.omitField(City.class, "description");[在生成的XML文档中省略City类的description]

<china>

<province name="辽宁省">

<city>

<name>沈阳</name>

</city>

<city>

<name>大连</name>

</city>

</province>

<province name="吉林省">

<city>

<name>长春</name>

</city>

<city>

<name>白城</name>

</city>

</province>

</china>