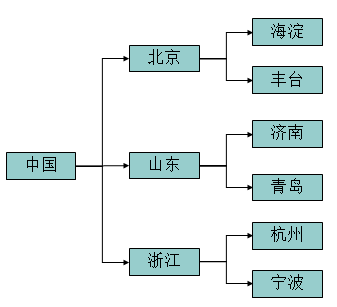
# XML

## 概述

什么是XML？

Extensible Markup Language，翻译过来为可扩展标记语言。Xml技术是w3c组织发布的，目前推荐遵循的是W3C组织于2000发布的XML1.0规范。

在现实生活中大量存在有关系的数据，如图所示。



问题：这样的数据如何交给计算机处理呢？

XML语言出现的根本目标在于描述上图那种，在现实生活中经常出现的有关系的数据。

在XML语言中，它允许用户自定义标签。一个标签用于描述一段数据；一个标签可分为开始标签和结束标签，在开始标签和结束标签之间，又可以使用其它标签描述其它数据，以此来实现数据关系的描述。例如：

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

<中国>

<北京>

<海淀></海淀>

<丰台></丰台>

</北京>

<山东>

<济南></济南>

<青岛></青岛>

</山东>

<浙江>

<杭州></杭州>

<宁波></宁波>

</浙江>

</中国>

XML技术除用于保存有关系的数据之外，它还经常用作软件配置文件，以描述程序模块之间的关系。

在一个软件系统中，为提高系统的灵活性，它所启动的模块通常由其配置文件决定

例如一个软件在启动时，它需要启动Ａ、Ｂ两个模块，而A、Ｂ这两个模块在启动时，又分别需要A1、A2和B1、B2模块的支持，为了准确描述这种关系，此时使用ＸＭＬ文件最为合适不过。

## XML语法

一个XML文件分为如下几部分内容：

文档声明

元素

属性

注释

CDATA区、特殊字符

处理指令（processing instruction）

### 文档声明：

在编写XML文档时，需要先使用文档声明，声明XML文档的类型。

最简单的声明语法：

<?xml version="1.0" ?>

用encoding属性说明文档的字符编码：

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

用standalone属性说明文档是否独立：

<?xml version="1.0" encoding="GB2312" standalone="yes" ?>

如果没有文档声明称为格式非良好的文档

### 元素

**XML元素指XML文件中出现的标签，一个标签分为开始标签和结束标签，一个标签有如下几种书写形式，例如：**

* + 包含标签体：<a>www.guigu.cn</a>
  + 不含标签体的：<a></a>, 简写为：<a/>

**一个标签中也可以嵌套若干子标签。但所有标签必须合理的嵌套，绝对不允许交叉嵌套，例如：**

<a>welcome to <b>www.it315.org</a></b>

**格式良好的XML文档必须有且仅有一个根标签，其它标签都是这个根标签的子孙标签。**

**对于XML标签中出现的所有空格和换行，XML解析程序都会当作标签内容进行处理。例如：下面两段内容的意义是不一样的。**

第一段：

<网址>www.guigu.cn</网址

第二段：

<网址>

www.guigu.cn

</网址>

**由于在XML中，空格和换行都作为原始内容被处理，所以，在编写XML文件时，使用换行和缩进等方式来让原文件中的内容清晰可读的“良好”书写习惯可能要被迫改变。**

**一个XML元素可以包含字母、数字以及其它一些可见字符，但必须遵守下面的一些规范：**

* + **区分大小写，例如，<P>和<p>是两个不同的标记。**
  + **不能以数字或"\_" (下划线)开头。**
  + **不能以xml(或XML、或Xml 等)开头。**
  + **不能包含空格。**
  + **名称中间不能包含冒号（:）。**

### 属性

**一个标签可以有多个属性，每个属性都有它自己的名称和取值，例如：**

<input name=“text”>

**属性值一定要用双引号（**"**）或单引号（'）引起来**

**定义属性必须遵循与标签相同的命名规范**

**在XML技术中，标签属性所代表的信息，也可以被改成用子元素的形式来描述，例如：**

<input>

<name>text</name>

</input>

### 注释

**Xml文件中的注释采用：“<!--注释-->” 格式。**

**注意：**

* + **XML声明之前不能有注释**
  + **注释不能嵌套，例如：**

<!--大段注释

……

<!--局部注释-->

……

-->

### CDATA区

* 在编写XML文件时，有些内容可能不想让解析引擎解析执行，而是当作原始内容处理。
* 遇到此种情况，可以把这些内容放在CDATA区里，对于CDATA区域内的内容，XML解析程序不会处理，而是直接原封不动的输出。
* 语法：<![CDATA[ 内容]]>

<![CDATA[

<guigu>

<br/>

</guigu>

]]>

### 转义字符



### 处理指令

**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。**

**例如，在XML文档中可以使用xml-stylesheet指令，通知XML解析引擎，应用css文件显示xml文档内容。** <?xml-stylesheet type="text/css" href="1.css"?>

**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。**

## XML约束

什么是XML约束？

在XML技术里，可以编写一个文档来约束一个XML文档的书写规范，这称之为XML约束。

为什么需要xml约束？

在项目中框架启动哪几个模块 需要读取配置通过xml来完成。如果不加以约束开发者随便定义，解析出错 ，框架中的内容无法使用，所以使用约束，框架设计者不能写一个文本去介绍， 而是通过约束、

常用的约束技术

XML DTD

XML Schema

### DTD

DTD(Document Type Definition)，全称为文档类型定义。

book.dtd

<!ELEMENT 书架 (书+)>

<!ELEMENT 书 (书名,作者,售价)>

<!ELEMENT 书名 (#PCDATA)>

<!ELEMENT 作者 (#PCDATA)>

<!ELEMENT 售价 (#PCDATA)>

+代表书可以出现一次或多次

#PCDATA字符串

book.xml

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

**<!DOCTYPE 书架 SYSTEM "book.dtd">**

<书架>

<书 name="xxxx">

<售价>39.00元</售价>

<书名>Java</书名>

<作者>李四</作者>

</书>

<书>

<书名>JavaScript网页开发</书名>

<作者>张三</作者>

<售价>28.00元</售价>

</书>

</书架>

#### 编程校验XML文档正确性：

把上面的改为 书1标签 ie仍然是正常是因为没有读取加载dtd

IE5以上浏览器内置了XML解析工具：Microsort.XMLDOM，开发人员可以编写javascript代码，利用这个解析工具装载xml文件，并对xml文件进行dtd验证。

创建xml文档解析器对象

var xmldoc = new ActiveXObject("Microsoft.XMLDOM");

开启xml校验

xmldoc.validateOnParse = "true";

装载xml文档

xmldoc.load("book.xml");

获取错误信息

xmldoc.parseError.reason;

xmldoc.parseError.line

编写一个html校验xml是否符合dtd的规范

<html>

<head>

<script>

var xmldoc = new ActiveXObject("Microsoft.XMLDOM");

xmldoc.validateOnParse=true;

xmldoc.load("config.xml");

document.write("错误的原因：" + xmldoc.parseError.reason + "<br/>");

document.write("错误的行号：" + xmldoc.parseError.line + "<br/>");

</script>

</head>

<body>

</body>

</html>

#### 引入dtd的两种方式：

可以单独一个文件 也可以放置在同一个文件内

如果约束和文件不是同一个文件 那么引入dtd的约束的方式又分为两种

* **XML文件使用DOCTYPE 声明语句来指明它所遵循的DTD文件，DOCTYPE声明语句有两种形式：**
  + **当引用的文件在本地时，采用如下方式：**

<!DOCTYPE文档根结点SYSTEM "DTD文件的URL">

例如：<!DOCTYPE书架SYSTEM “book.dtd”>。**在xml文件中手写一下。**

* + **当引用的文件是一个公共的文件时，采用如下方式：**

<!DOCTYPE文档根结点PUBLIC "DTD名称""DTD文件的URL">

**例如：**

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

#### 元素定义

* 在DTD文档中使用ELEMENT声明一个XML元素，语法格式如下所示：

<!ELEMENT 元素名称 元素类型>

* 元素类型可以是元素内容、或类型
  + 如为元素内容：则需要使用()括起来，如

<!ELEMENT书架(书名，作者，售价)>

<!ELEMENT书名(#PCDATA)>

* + 如为元素类型，则直接书写，DTD规范定义了如下几种类型：
    - EMPTY：用于定义空元素，例如<br/><hr/>
    - ANY：表示元素内容为任意类型。

<!ELEMENT书架 ANY>

<!ELEMENT书 (书名,作者,售价)>

<!ELEMENT书名 (#PCDATA)>

<!ELEMENT作者 (#PCDATA)>

<!ELEMENT售价 EMPTY>

* 元素内容中可以使用如下方式，描述内容的组成关系
  + 用逗号分隔，表示内容的出现顺序必须与声明时一致。

<!ELEMENT MYFILE (TITLE,AUTHOR,EMAIL)>

* + 用|分隔，表示任选其一，即多个只能出现一个

<!ELEMENT MYFILE (TITLE|AUTHOR|EMAIL)>

* + 元素内容使用|分隔，后面再加上\*，表示出现顺序没有要求：<!ELEMENT MYFILE (TITLE |AUTHOR|EMAIL)\*>
* 在元素内容中也可以使用+、\*、?等符号表示元素出现的次数：

+: 一次或多次(书+)

?: 0次或一次(书?)

\*: 0次或多次(书\*)

* 也可使用圆括号( )批量设置，例

<!ELEMENT MYFILE ((TITLE\*, AUTHOR?, EMAIL)\* | COMMENT)>

#### 属性定义

* **xml文档中的标签属性需通过ATTLIST为其设置属性**
* **语法格式：**

<!ATTLIST元素名

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

……

>

* **属性声明举例：**

<!ATTLIST商品

类别 CDATA #REQUIRED

颜色 CDATA #IMPLIED

>

REQUIRED必选的

IMPLIED可选的

* **对应XML文件：**

<商品 类别="服装" 颜色="黄色">…</商品>

<商品 类别="服装">…</商品>

* **设置说明：**
  + #REQUIRED：必须设置该属性
  + #IMPLIED：可以设置也可以不设置
  + #FIXED：说明该属性的取值固定为一个值，在 XML 文件中不能为该属性设置其它值。但需要为该属性提供这个值
  + 直接使用默认值：””在 XML 中可以设置该值也可以不设置该属性值。若没设置则使用默认值。
* **举例：**

**<!ATTLIST**页面作者

姓名 CDATA #IMPLIED

年龄 CDATA #IMPLIED

联系信息 CDATA #REQUIRED

网站职务 CDATA #FIXED "页面作者"

个人爱好 CDATA "上网"

**>**

属性值的类型：

1，CDATA

,2，属性的类型可以是一组取值的列表，在XML 文件中设置的属性值只能是这个列表中的某个值(枚举)

**<?xml version = "1.0" encoding="GB2312" standalone="yes"?>**

**<!DOCTYPE购物篮 [**

**<!ELEMENT 肉 EMPTY>**

**<!ATTLIST 肉 品种 ( 鸡肉 | 牛肉 | 猪肉 | 鱼肉 ) "鸡肉">**

**]>**

**<购物篮>**

**<肉 品种="鱼肉"/>**

**<肉 品种="牛肉"/>**

**<肉/>**

**</购物篮>**

3，ID ：

* 表示属性的设置值为一个唯一值。
* ID 属性的值只能由字母，下划线开始，不能出现空白字符

?xml version = "1.0" encoding="GB2312" ?>

<!DOCTYPE联系人列表[

<!ELEMENT联系人列表 ANY>

<!ELEMENT联系人(姓名,EMAIL)>

<!ELEMENT姓名(#PCDATA)>

<!ELEMENT EMAIL(#PCDATA)>

<!ATTLIST 联系人 编号 ID #REQUIRED>

]>

<联系人列表>

<联系人 编号="a1">

<姓名>张三</姓名>

<EMAIL>zhang@it315.org</EMAIL>

</联系人>

<联系人编号="b1">

<姓名>李四</姓名>

<EMAIL>li@it315.org</EMAIL>

</联系人>

</联系人列表>

4,entity(实体)：

实体用于为一段内容创建一个别名，以后在XML文档中就可以使用别名引用这段内容了。

在DTD定义中，一条<!ENTITY …>语句用于定义一个实体。

实体可分为两种类型：引用实体和参数实体。

引用实体主要在 XML 文档中被应用

语法格式：

<!ENTITY 实体名称 “实体内容”>：直接转变成实体内容

引用方式：(注意 后面有分号)

&实体名称;

举例：

<!ELEMENT书架 ANY>

<!ELEMENT书 (书名,作者,售价)>

<!ELEMENT书名 (#PCDATA)>

<!ELEMENT作者 (#PCDATA)>

<!ELEMENT售价 EMPTY>

<!ENTITY book “java web”>

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

<!DOCTYPE书架 SYSTEM "book.dtd">

<书架>

<书 name="xxxx">

<售价>39.00元</售价>

<书名>&book;</书名>

<作者>李四</作者>

</书架>

参数实体被 DTD 文件自身使用

语法格式：

<!ENTITY % 实体名称 "实体内容">

引用方式：

%实体名称;

举例1：

<!ENTITY % TAG\_NAMES "姓名 | EMAIL | 电话 | 地址">

<!ELEMENT个人信息 (%TAG\_NAMES; | 生日)>

<!ELEMENT客户信息 (%TAG\_NAMES; | 公司名)>

举例2：

<!ENTITY % common.attributes

" id ID #IMPLIED

account CDATA #REQUIRED "

>

...

<!ATTLIST purchaseOrder %common.attributes;>

<!ATTLIST item %common.attributes;>

为标签purchaseOrder 声明 属性

## Schema

* XML Schema 也是一种用于定义和描述 XML 文档结构与内容的模式语言，其出现是为了克服 DTD 的局限性
* XML Schema VS DTD：
  + XML Schema符合XML语法结构。
  + DOM、SAX等XML API很容易解析出XML Schema文档中的内容。
  + XML Schema对**名称空间**支持得非常好。
  + XML Schema比XML DTD支持更多的数据类型，并支持用户自定义新的数据类型。
  + XML Schema定义约束的能力非常强大，可以对XML实例文档作出细致的语义限制。
  + XML Schema不能像DTD一样定义实体，比DTD更复杂，但Xml Schema现在已是w3c组织的标准，它正逐步取代DTD。

### 快速入门

* XML Schema 文件自身就是一个XML文件，但它的扩展名通常为.xsd。
* 一个XML Schema文档通常称之为**模式文档**(约束文档)，遵循这个文档书写的xml文件称之为**实例文档**。
* 和XML文件一样，一个XML Schema文档也必须有一个根结点，但这个根结点的名称为Schema。
* 编写了一个XML Schema约束文档后，通常需要把这个文件中声明的元素绑定到一个ＵＲＩ地址上，在XML Schema技术中有一个专业术语来描述这个过程，即把XML Schema文档声明的元素绑定到一个**名称空间**上，以后XML文件就可以通过这个URI（即名称空间）来告诉解析引擎，xml文档中编写的元素来自哪里，被谁约束。

案例：book.xsd文件

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

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

targetNamespace="http://www. guigu.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:string' />

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

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

<guigu:书架 xmlns:guigu="http://www.guigu.cn"

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

xsi:schemaLocation=“http://www.guigu.cn book.xsd">

<guigu:书>

<guigu:书名>JavaScript网页开发</guigu:书名>

<guigu:作者>张三</guigu:作者>

<guigu:售价>28.00元</guigu:售价>

</guigu:书>

</guigu:书架>

### 名称空间

* 在XML Schema中，每个约束模式文档都可以被赋以一个唯一的名称空间，名称空间用一个唯一的URI（Uniform Resource Identifier，统一资源标识符）表示。 在Xml文件中书写标签时，可以通过名称空间声明（xmlns），来声明当前编写的标签来自哪个Schema约束文档。如：

<guigu:书架 xmlns:guigu=“http://www.guigu.cn”>

<guigu:书>……</guigu:书>

</guigu:书架>

**此处使用guigu来指向声明的名称，以便于后面对名称空间的引用。**

**使用名词空间引入schema：**

* 为了在一个XML文档中声明它所遵循的Schema文件的具体位置，通常需要在Xml文档中的根结点中使用schemaLocation属性来指定，例如：

<guigu:书架 xmlns:guigu="http://www.guigu.cn"

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

xsi:schemaLocation=“http://www.guigu.cn book.xsd">

* schemaLocation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，两者之间用空格分隔。
* 注意，在使用schemaLocation属性时，也需要指定该属性来自哪里。

使用默认的名称空间：

* **基本格式：**

xmlns="URI"

* **举例：**

<书架 xmlns="http://www.it315.org/xmlbook/schema"

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

xsi:schemaLocation=“http://www.guigu.cn book.xsd">

<书>

<书名>JavaScript网页开发</书名>

<作者>张三</作者>

<售价>28.00元</售价>

</书>

<书架>

使用名称空间引入多个XML Schema文档:

**文件清单：xmlbook.xml**

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

<书架 xmlns="http://www.it315.org/xmlbook/schema"

xmlns:demo="http://www.it315.org/demo/schema"

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

xsi:schemaLocation="http://www.it315.org/xmlbook/schema http://www.it315.org/xmlbook.xsd

http://www.it315.org/demo/schema http://www.it315.org/demo.xsd">

<书>

<书名>JavaScript网页开发</书名>

<作者>张三</作者>

<售价 demo:币种=”人民币”>28.00元</售价>

</书>

</书架>

不使用名称空间引入XML Schema文档:

**文件清单：xmlbook.xml**

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

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

xsi:noNamespaceSchemaLocation="xmlbook.xsd">

<书>

<书名>JavaScript网页开发</书名>

<作者>张三</作者>

<售价>28.00元</售价>

</书>

</书架>

在XML Schema文档中声明名称空间:

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

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

elementFormDefault="qualified">

<xs:schema>

* targetNamespace元素用于指定schema文档中声明的元素属于哪个名称空间。

elementFormDefault元素用于指定，该schema文档中声明的根元素及其所有子元素都属于targetNamespace所指定的名称空间

### 通过[www.w3school.com.cn](http://www.w3school.com.cn/)学习schema

## XML编程

* XML解析方式分为两种：dom和sax
  + dom：(Document Object Model, 即文档对象模型) 是W3C 组织推荐的处理XML 的一种方式。
  + sax：(Simple API for XML) 不是官方标准，但它是XML 社区事实上的标准，几乎所有的XML 解析器都支持它。
* XML解析器

Crimson（sun）、Xerces（ibm）、Aelfred2（dom4j）

* XML解析开发包
  + Jaxp、Jdom、dom4j

### Jaxp操作XML 增删改查dom

JAXP 开发包是J2SE的一部分，它由javax.xml、org.w3c.dom 、org.xml.sax 包及其子包组成

在 javax.xml.parsers 包中，定义了几个工厂类，程序员调用这些工厂类，可以得到对xml文档进行解析的 DOM 或 SAX 的解析器对象。

javax.xml.parsers 包中的DocumentBuilderFactory用于创建DOM模式的解析器对象 ， DocumentBuilderFactory是一个抽象工厂类，它不能直接实例化，但该类提供了一个newInstance方法 ，这个方法会根据本地平台默认安装的解析器，自动创建一个工厂的对象并返回。

获得JAXP中的DOM解析器:

调用 DocumentBuilderFactory.newInstance() 方法得到创建 DOM 解析器的工厂。

调用工厂对象的 newDocumentBuilder方法得到 DOM 解析器对象。

调用 DOM 解析器对象的 parse() 方法解析 XML 文档，得到代表整个文档的 Document 对象，进行可以利用DOM特性对整个XML文档进行操作了。

我们把book1.xml放到src下面，然后利用jaxp来对它进行操作：

book.xml

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

<书架>

<书 name="xxxx">

<售价>39.00元</售价>

<书名>Java</书名>

<作者>李四</作者>

</书>

<书>

<书名>JavaScript网页开发</书名>

<作者>张三</作者>

<售价>28.00元</售价>

</书>

</书架>

### jaxp解析XML文档sax 查

在使用 DOM 解析 XML 文档时，需要读取整个 XML 文档，在内存中构架代表整个 DOM 树的Doucment对象，从而再对XML文档进行操作。此种情况下，如果 XML 文档特别大，就会消耗计算机的大量内存，并且容易导致内存溢出。

SAX解析允许在读取文档的时候，即对文档进行处理，而不必等到整个文档装载完才会文档进行操作。边读取边解析边操作

使用 创建SAX解析工厂

SAXParserFactoryspf = SAXParserFactory.newInstance();

通过SAX解析工厂得到解析器对象

SAXParser sp = spf.newSAXParser();

通过解析器对象得到一个XML的读取器

XMLReaderxmlReader = sp.getXMLReader();

设置读取器的事件处理器

xmlReader.setContentHandler(new BookParserHandler());

解析xml文件

xmlReader.parse("book.xml");

### DOM4J解析XML文档

导包dom4j

#### DOM4j中，获得Document对象的方式有三种：

1.读取XML文件,获得document对象

SAXReader reader = new SAXReader();  
              Document   document = reader.read(new File("input.xml"));

2.解析XML形式的文本,得到document对象.

      String text = "<members></members>";  
              Document document = DocumentHelper.parseText(text);

3.主动创建document对象.

             Document document = DocumentHelper.createDocument();  
             //创建根节点

Element root = document.addElement("members");

#### 节点对象：

1.获取文档的根节点.

Element root = document.getRootElement();

2.取得某个节点的子节点.

Element element=node.elements(“书名").get(2);

Element element=node.element(“书名")

3.取得节点的文字

String text=node.getText();

4.取得某节点下所有名为“member”的子节点，并进行遍历.  
 List nodes = rootElm.elements("member");  
   
 for (Iterator it = nodes.iterator(); it.hasNext();) {  
     Element elm = (Element) it.next();  
    // do something  
 }

5.对某节点下的所有子节点进行遍历.  
  for(Iterator it=root.elementIterator();it.hasNext();){  
       Element element = (Element) it.next();  
       // do something  
    }

6.在某节点下添加子节点.  
Element ageElm = newMemberElm.addElement("age");

7.设置节点文字.  
 element.setText("29");

8.删除某节点.  
//childElm是待删除的节点,parentElm是其父节点

parentElm.remove(childElm);

9.添加一个CDATA节点.  
Element contentElm = infoElm.addElement("content");  
contentElm.addCDATA(diary.getContent());

#### 节点对象属性：

1.取得某节点下的某属性  
    Element root=document.getRootElement();      
    //属性名name

Attribute attribute=root.attribute("size");

2.取得属性的文字  
    String text=attribute.getText();

3.删除某属性  
 Attribute attribute=root.attribute("size");  
 root.remove(attribute);

3.遍历某节点的所有属性  
   Element root=document.getRootElement();      
  for(Iterator it=root.attributeIterator();it.hasNext();){  
         Attribute attribute = (Attribute) it.next();  
         String text=attribute.getText();  
         System.out.println(text);  
    }

4.设置某节点的属性和文字.  
 newMemberElm.addAttribute("name", "sitinspring");

5.设置属性的文字  
 Attribute attribute=root.attribute("name");  
   attribute.setText("sitinspring");

#### 将内存中的文档写入xml文件：

1.文档中全为英文,不设置编码,直接写入的形式.  
 XMLWriter writer = new XMLWriter(new FileWriter("output.xml"));  
 writer.write(document);  
 writer.close();

2.文档中含有中文,设置编码格式写入的形式.  
OutputFormat format = OutputFormat.createPrettyPrint();  
// 指定XML编码

format.setEncoding("utf-8");         
XMLWriter writer = **new** XMLWriter(**new** FileOutputStream("book.xml"), format);  
writer.write(document);  
writer.close();

#### 7 在指定的位置插入节点：

1.得到插入位置的节点列表（list）

2.调用list.add(index,elemnent)，由index决定element的插入位置。

Element元素可以通过DocumentHelper对象得到。示例代码：

Element aaa = DocumentHelper.createElement("aaa");

aaa.setText("aaa");

List list = root.element("书").elements();

list.add(1, aaa);

//更新document

#### 字符串与xml的转换：

##### 1.将字符串转化为XML

String text = "<members> <member>sitinspring</member></members>";  
Document document = DocumentHelper.parseText(text);

##### 2.将文档或节点的XML转化为字符串.

#### Xpath

通过XPathTutorial查阅