# 一、XML概述

英文全称为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，目前遵循的是W3C组织于2000发布的XML1.0规范。

现实生活中存在着大量的数据，在这些数据之间往往存在一定的关系，我们希望能在计算机中保存和处理这些数据的同时能够保存和处理他们之间的关系。

XML就是为了解决这样的需求而产生数据存储格式。

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

<中国>

<北京>

<海淀></海淀>

<丰台></丰台>

</北京>

<河北>

<唐山></唐山>

<石家庄></石家庄>

</河北>

<山东>

<济南></济南>

<青岛></青岛>

</山东>

。。。。

</中国>

在XML语言中，它允许用户自定义标签。每一个标签用于描述一段数据；

一个标签可以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嵌套其它标签，利用标签间的嵌套关系来保存数据之间的上下级关系；

由于xml实质上是一段字符串，计算机可以十分方便的对他进行操作，开发人员也可以方便的阅读，因此可以说这是一种对人、对计算机都友好的数据存储格式，所以XML迅速普及，成为了一种非常常见的数据存储格式，在许多应用场景中得到应用。

XML本质上是一段字符串，具有跨平台性的特性，因此XML常被用来在不同系统之间进行数据交换。

XML可以在保存数据的同时保存数据之间的关系。利用这一特点，它还经常用作应用程序配置文件来使用

XML文件：将符合xml语法的数据存储到一个文件中（这样的文件通常后缀名起为.xml）就得到了一个存储着xml数据的 xml文件

XML校验：浏览器除了可以处理和展示html格式的数据以外也可以处理xml格式的数据，所以利用浏览器打开xml格式的数据，如果正确就会正常显示，如果不正确浏览器识别出来会有提示。

# 二、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）

1.文档声明

文档声明通常是用来声明xml的基本属性的，xml的文档声明必须放置在整个xml文件的第一行，有且仅有一个，之前不能有任何其他内容。

第一种写法：version用来声明当前xml遵循的是哪个版本的xml

<?xml version="1.0" ?>

第二种写法：encoding属性用来通知解析器当前xml采用的编码，利用这个头可以解决xml文件的乱码问题

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

第三种写法：standalone属性用来声明当前xml文档是否是一个独立文档。可以取值为yes表示是一个独立文档，也可以取值位no表示不是一个独立文档，该文档需要依赖其他文档而存在。默认值是yes。

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

2.元素

一个标签就是一个xml元素

标签分为开始标签和结束标签，开始标签和结束标签之间包含的文本称为该标签的表前体

标签内部也可以嵌套其他标签，但是要注意一定要合理嵌套，不能交叉嵌套

如果一个标签内部既没有标签体也没有任何子标签，则该标签可以将开始标签和结束标签进行合并称为自闭标签 <a></a> --> <a/>

一个格式良好的xml文件应该有且仅有一个根标签，其他所有的标签都应该是这个根标签的子孙标签。

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。

不能以数字或标点符号或"\_"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（:）。

3.属性

一个xml元素可以具有零个或多个属性

属性是由属性名和属性值组成的，属性名和属性值之间用=连接，属性的值应该用双引号或单引号引起来

定义属性名必须遵循与元素相同的命名规范

4.注释

<!--注释的内容-->

注释不能嵌套注释

注释不能出现在文档声明之前,因为文档声明必须处在整个文档的最前面，之前不能有任何内容包括注释！

## 5.CDATA区 转义字符

转义字符：如果xml文档中的普通字符内容中包含了一些特殊字符导致解析引擎解析出问题，可以通过转义字符进行转移

< --> &lt;

> --> &gt;

& --> &amp;

" --> &quot;

‘ --> &apos;

转义字符适用于需要转义的内容比较少的情况，如果需要将一大段文本中的特殊内容进行转义，可以使用CDATA区

CDATA区：

<![CDATA[

需要转义的内容

]]>

CDATA区适用于将一大段内容进行转义，好处是可以一次处理一大段内容，不需要挨个处理其中的每个特殊字符， 另外使用CDATA区不会改变原文内容。

## 6.处理指令：

xml中的处理指令用来通知解析引擎如何来解析xml中的其他部分内容

<?处理指令的名称 若干属性 ?>

<?xml-stylesheet href="" type=""?>

# 三、xml约束 -- 要求看的懂约束文件 不要求会写

一个xml的写法可以用另外一个文件来进行约束，这样的技术称为xml的约束技术。

xml的约束技术可以用来约束xml的写法

写出xml文件也可以通过xml的约束文件进行校验

dtd 语法简单 容易学习 但是功能比较弱

schema 语法强大 可以实现语意级别的约束 语法复杂 学习成本高

## 1.DTD

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

案例：

<!ELEMENT 书架 (书+)>

<!ELEMENT 书 (书名,作者,售价)>

<!ELEMENT 书名 (#PCDATA)>

<!ELEMENT 作者 (#PCDATA)>

<!ELEMENT 售价 (#PCDATA)>

\*\*\*注意，DTD文件必须是UTF-8编码，否则无法处理

<?xml version="1.0" encoding="utf-8" standalone="no" ?>

<!DOCTYPE 书架 SYSTEM "book.dtd">

<书架>

<书>

<书名>mysql编程指南</书名>

<作者>郭德纲</作者>

<售价>9.0元</售价>

</书>

<书>

<书名>JavaScript入门案例</书名>

<作者>萨达姆</作者>

<售价>8.0元</售价>

</书>

</书架>

\*\*\*默认情况下 浏览器并没有开启DTD校验，所以不能是用浏览器来教研xml是否符合dtd的约束 可以通过JavaScript语法利用代码进行dtd校验。

\*\*\*其实，还有更简单的方法，我们的eclipse或myeclipse都能自动进行dtd教研的工作

在xml中引入DTD：

引入外部文件：

可以在外部文件中写好dtd，在xml内部通过标签进行引入操作

引入本地文件：

<!DOCTYPE 根元素的名称 SYSTEM "dtd文件位置">

引入网络公共位置文件：

<!DOCTYPE 根元素的名称 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">

在xml文件内部定义：

在xml的文档声明后通过语法直接将DTD写在xml中

<!DOCTYPE 根元素名称 [

dtd内容

]>

DTD语法：

元素声明

<!ELEMENT 元素名称 元素的约束>

元素的约束：

存放类型、子元素的列表

存放类型：

EMPTY 表示该元素内部不能有任何其他内容

ANY 表示该元素内部可以是任意内容

子元素的列表：表示该元素内部按要求出现子元素

子元素需要用小括号引起来，对于标签体用#PCDATA表示

子元素之间可以使用逗号","进行分割，表示子元素必须按顺序出现

子元素之间也可以使用竖线"|"进行分割，表示子元素只能出现其中之一

另外可以使用数量词描述子元素出现的次数：

+：表示出现1次或多次

\*：表示出现0次或多次

?：表示出现0次或1次

还可以使用小括号进行组的操作

例如：

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

属性声明

<!ATTLIST 元素名

属性名1 属性类型 属性约束

属性名2 属性类型 属性约束

……

>

属性类型

CDATA 表示属性是一个普通的值

ENUMERATED 表示属性取值是在一个给定的枚举范围内

ID 表示该属性作为ID来使用，ID的特定是在整个文档范围内唯一，ID的值不能以数字开头

ENTITY(实体)

属性约束：

#REQUIRED 必须存在的属性

#IMPLIED 可选的属性

#FIXED 固定值 表示具有一个固定值的属性，可以不设定该属性，概述就有且取值为给定的固定值，如果设定了且设定的值和给定的固定值不同，则报错。

默认值 表示该属性具有一个默认值，如果设置该属性，该属性也有取值就位这个默认值，如果设置了该属性，取值就是给定的值

实体声明

所谓的实体就是将一大段的文本设定位实体，让其具有一个引用，在需要的地方可以通过引用来使用这个实体中定义的大段文本

引用实体：在dtd中定义在xml中使用的实体

<!ENTITY 实体的名称 实体的内容>

在xml中通过 &实体名称; 使用实体

参数实体：在dtd中定义在dtd中使用的实体

<!ENTITY % 实体名称 实体内容>

在dtd中通过 %实体名称; 使用实体

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)>

2.schema

# 四、xml解析

利用java程序操作xml中的数据

## 1.两种解析思想

dom方式解析

sax方式的解析

2.解析api

sun --> jaxp 原生的加入了j2se技术中 既有dom方式 也有sax方式 不需要导入其他开发包直接就可以使用

!!dom4j --> 开源组织提供的解析api api简单 性能高效 很多地方都有所应用 内部集成了sax和dom两种方式。

jdom

pull --> sax解析方式的api 效率高 api简单

。。。

3.dom4j

下载并导入开发包

利用dom4j实现对xml的增删改查（Create Read Update Delete）

|  |
| --- |
| public void attr() throws Exception{  //1.解析xml获取dom  SAXReader reader = new SAXReader();  Document dom = reader.read("book.xml");  Element root = dom.getRootElement();  //2.获取父节点  Element bookEle = root.element("书");    //--增加属性方式1  //凭空创建属性对象  //Attribute attr = DocumentHelper.createAttribute(bookEle, "出版社", "人民出版社");  //挂载上去  //bookEle.add(attr);    //---新增属性方法2  //bookEle.setAttributeValue("种类", "工具书");    //--修改属性  //bookEle.setAttributeValue("种类", "文学书");    //--查找属性值方式1  //Attribute attr = bookEle.attribute("出版社");  //System.out.println(attr.getName());  //System.out.println(attr.getValue());    //--查找属性值方式2  //String v = bookEle.attributeValue("出版社");  //System.out.println(v);    //--删除属性  //Attribute attr = bookEle.attribute("出版社");  //bookEle.remove(attr);    //5.更新xml  XMLWriter writer = new XMLWriter(new FileOutputStream("book.xml"),OutputFormat.createCompactFormat().createPrettyPrint());  writer.write(dom);  writer.close();    }    @Test  public void add2() throws Exception{  //1.解析xml获取dom  SAXReader reader = new SAXReader();  Document dom = reader.read("book.xml");  Element root = dom.getRootElement();  //2.获取父节点  Element bookEle = root.element("书");  //3.凭空创建出节点  Element price2Ele = DocumentHelper.createElement("特价");  price2Ele.setText("0.5元");  //4.挂载  List<Element> list = bookEle.elements();  list.add(1, price2Ele);  //5.写回  XMLWriter writer = new XMLWriter(new FileOutputStream("book.xml"),OutputFormat.createCompactFormat().createPrettyPrint());  writer.write(dom);  writer.close();  }    @Test  public void del() throws Exception{  //1.解析xml获取dom  SAXReader reader = new SAXReader();  Document dom = reader.read("book.xml");  Element root = dom.getRootElement();  //2.获取要删除的节点  Element price2Ele = root.element("书").element("特价");  //3.获取要删除节点的父节点,删除节点  price2Ele.getParent().remove(price2Ele);  //4.写出到文件中  XMLWriter writer = new XMLWriter(new FileOutputStream("book.xml"),OutputFormat.createCompactFormat().createPrettyPrint());  writer.write(dom);  writer.close();  }    @Test  public void find() throws Exception{  //1.解析xml获取dom  SAXReader reader = new SAXReader();  Document dom = reader.read("book.xml");  Element root = dom.getRootElement();  //2.获取第二本书的名字  List<Element> list = root.elements("书");  Element bookEle = list.get(1);  Element bookNameEle = bookEle.element("书名");  String bookName = bookNameEle.getText();  System.out.println(bookName);  }    @Test  public void update() throws Exception{  //1.解析xml获取dom  SAXReader reader = new SAXReader();  Document dom = reader.read("book.xml");  Element root = dom.getRootElement();  //2.获取需要修改的节点  Element price2Ele = root.element("书").element("特价");  //3.修改内容  price2Ele.setText("1.0元");  //4.写回文件中  XMLWriter writer = new XMLWriter(new FileOutputStream("book.xml"),OutputFormat.createCompactFormat().createPrettyPrint());  writer.write(dom);  writer.close();  }    @Test  public void add() throws Exception{  //1.解析xml获取dom  SAXReader reader = new SAXReader();  Document dom = reader.read("book.xml");  Element root = dom.getRootElement();  //2.凭空创建出<特价>节点  Element price2Ele = DocumentHelper.createElement("特价");  price2Ele.setText("0.9元");  //3.找到父节点  Element bookEle = root.element("书");  //4.挂载上去  bookEle.add(price2Ele);  //5.将dom写回到文件中  // Writer writer = new OutputStreamWriter(new FileOutputStream("book.xml"),"utf-8");  // dom.write(writer);  // writer.close();    XMLWriter writer = new XMLWriter(new FileOutputStream("book.xml"),OutputFormat.createPrettyPrint());  writer.write(dom);  writer.close();    } |

# 五、schema

## 名称空间（命名空间） -- Namespace

其实就是一个名字，特点是全世界独一无二的名字，为了保证这种独一无二的特性，一般会用公司的域名进行变换，由于域名时独一无二的，这样的名称空间，就可以保证时全世界独一无二的。

===========================================================

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

Schema与 DTD的比较：

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。

===========================================================

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。

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

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

targetNamespace=“www.tarena.com"

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"?>

<tarena:书架 xmlns:tarena=“www.tarena.com”

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

xsi:schemaLocation="“www.tarena.com book.xsd">

<tarena:书>

<tarena:书名>Java开发指南</tarena:书名>

<tarena:作者>萨达姆</tarena:作者>

<tarena:售价>9.00元</tarena:售价>

</tarena:书>

</tarena:书架>

===========================================================

## 名称空间的概念

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：

<tarena:书架 xmlns:tarena=“http://www.tarena.com”>

<tarena:书>……</tarena:书>

</tarena:书架>

此处使用tarena来指向声明的名称，以便于后面对名称空间的引用。

注意：名称空间的名字语法容易让人混淆，尽管以 http:// 开始，那个 URL 并不指向一个包含模式定义的文件。事实上，这个 URL：http://www.tarena.com根本没有指向任何文件，只是一个分配的名字。

===========================================================

## XML中使用名称空间引入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：

<tarena:书架 xmlns:tarena="http://www.tarena.com"

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

xsi:schemaLocation=“http://www.tarena.com 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。

===========================================================

## 使用默认名称空间

基本格式：

xmlns="URI"

举例：

<书架 xmlns="http://www.tarena.com"

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

xsi:schemaLocation="“http://www.tarena.com book.xsd">

<书>

<书名>JavaScript开发</书名>

<作者>萨达姆</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.park.com/xmlbook/schema"

xmlns:demo="http://www.park.com/demo/schema"

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

xsi:schemaLocation="http://www.park.com/xmlbook/schema http://www.park.com/xmlbook.xsd

http://www.park.com/demo/schema http://www.park.com/demo.xsd">

<书>

<书名>JavaScript网页开发</书名>

<作者>萨达姆</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>萨达姆</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. tarena.com"

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是否都属于targetNamespace所指定的名称空间。

===========================================================

Schema语法

参看w3c文档

===========================================================