# XML的简介

## 可扩展标记型语言（extensible Markus language）

### 概述

XML是一种可扩展的标记语言，它被设计用来传输和存储数据，是由万维网协会推出的一套数据。它可以用于定义Web网页上的文档元素，以及复杂数据的表述和传输。XML与HTML的主要差异在于：HTML用来显示数据，XML用来传输和存储数据；HTML用来显示信息，XML用来传输信息。XML的最大特点是它的自我描述和任意扩展，当用其描述数据时，用户可以根据需要，组织符合XML规范形式的任意内容，并且标签的名称也可以由用户指定。XML是w3c组织发布的一个技术，版本 1.0 和1.1（使用的一般为1.0版本 1.1向下不兼容）

XML：可扩展标记型语言。

标记型语言：html标记型语言，也是使用标签来操作

可扩展：

html：里面的标签是固定的，每个标签都有特定的含义

XML：标签可以使自己定义，可以中文的标签。

例如：<person></person> <猫></猫>

### XML的用途：

1)传输数据

通过XML可以在不同的系统之间传输数据，在开发过程中难免会遇到多个系统之间相互通信，且各系统的存储数据又是多种多样的情况，对于开发者而言，这些工作量是巨大的，通过转换为XML格式来传输数据可以减少传输数据时的复杂性，并且还可以具备通用性。

2)存储数据(存储数据)

利用XML来存储数据是其最基本的用途，因为它可以作为数据文件，所以当需要持久化保存数据时，可以利用XML数据格式进行存储，例如，web.XML、struts.XML、spring.XML等。数据的最终位置是在数据库中。下面就是经常见到的web.XML文件内容：

3)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来实现不同数据库中数据的交换。将Oracle的数据导出一个XML文件，然后通过dom4j技术读取XML的文件内容，最后导入Mysql中。

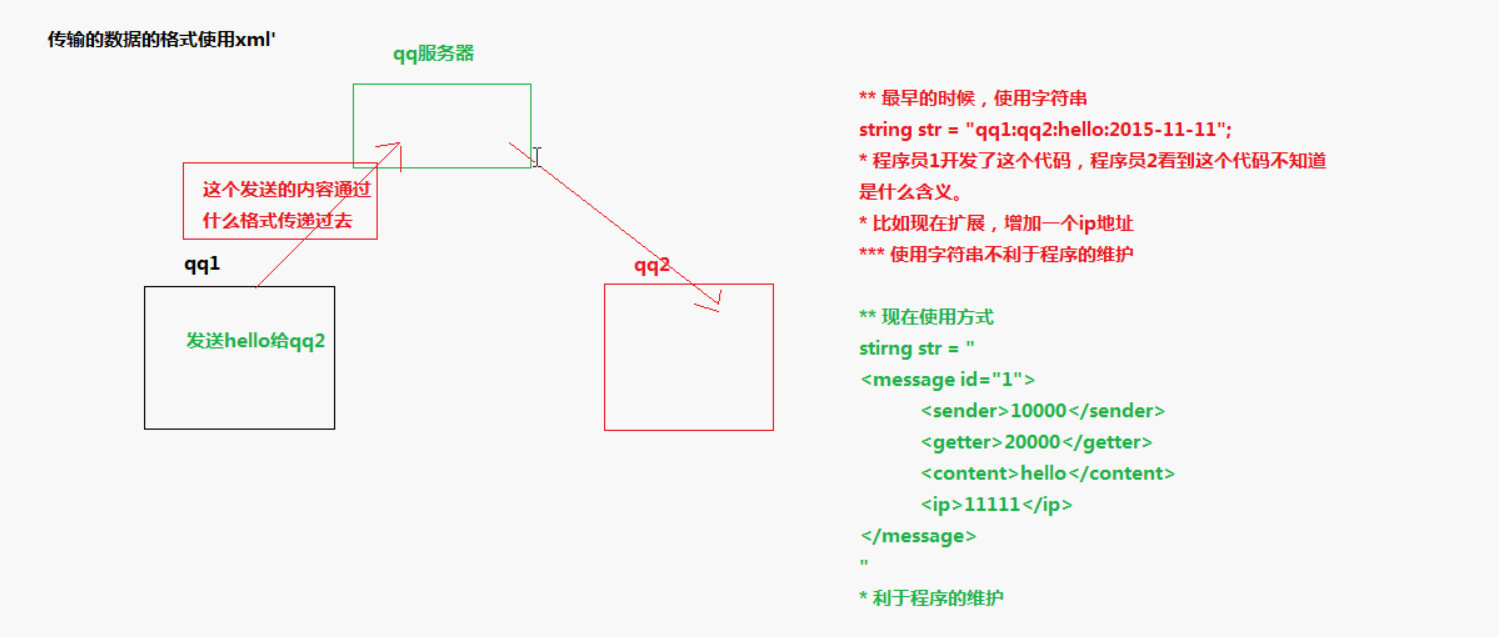
如何解决：将MySQL数据库一个表中的数据库导出到xml文件中，再通过dom4j读取

xml文件，将xml中的数据写入到Oracle数据库中。

### XML的应用

1、不同的系统之间传输数据

qq之间传输数据，例如：



2、用来表示生活中有关系的的数据

可以用XML来实现有关系的数据。

例如：山西省-太原市-迎泽区

|  |
| --- |
| <山西>  <太原>  <迎泽区>  </迎泽区>  </太原>  </山西> |

3、经常用在配置文件、

比如现在连接数据库：知道数据库的用户名和密码数据库名称

如果修改数据的信息，不需要修改源代码，只需要修改配置文件。

### XML的优点

1)简单，普通的文本文件，计算机可以直接识别。

2)良好的可读性(格式与内容分离)

3)可扩展性

4)可以轻松的跨平台应用。

5)数据内容与其形式分离。

## 数据的传输格式

### XML的技术架构

### XML的文档声明

创建一个文件后缀.XML，第一步必须写一个文档说明（写了文档声明之后，表示写XML里面的东西）

声明格式：<?xml version="1.0" encoding="UTF-8" standalone="no"?>

注意点：

1)位置：文档声明的位置必须为第一行第一列。

2)属性：version XML的标准版本 1.0 1.1 两个版本

Encodeing：指明文档使用的字符编码格式，XML编码

gbk中文简体

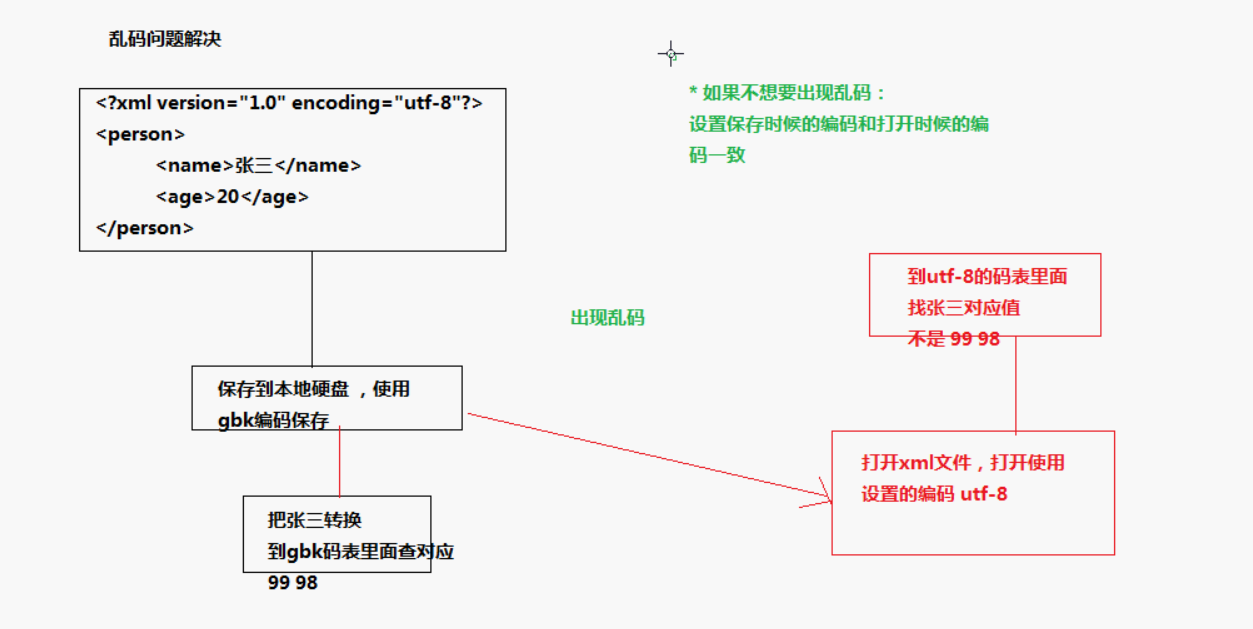
UTF-8

iso8859-1（不包含中文）。

Standalone：是否需要依赖其他的文件值 yes no。如果XML文档引用其它描述该文档可以包含什么的文件(如DTD)，则 standalone=”no”。默认值为”no”。

注：属性之间要用空格分隔。

### XML的中文乱码：



保存时候的编码和大家时候的编码要一致

### XML的元素的定义（标签的定义）

标签定义:<person></person>

注：

1)标签成对形式出现。

2)标签没有内容，可以在标签内结束<a/>。

3)标签可以嵌套，但是必须要合理嵌套。

合理嵌套：<dad><son></son></dad>即dad是son的父元素son是dad的子元素。

不合理嵌套：<aa><bb></aa></bb>不合理嵌套.

4)一个XML有且只有一个根标签，其他的标签都是这个标签这么下面的标签 .

5)在XML中，它会将空格和换行内容进行解析，下面中的代码含义是不一样的。

<aa>111111</aa>

<aa>//这里的会被解析。

1111

</aa>

### XML中标签的命名规则

1、XML代码中是区分大小写的。<age><Age>是两个不同的标签。

2、标签名称不能以数字和下划线开头，只能以字母和中文字开头。

3、元素的名字不能以xml（XML、Xml、xML等）开头。

4、XML标签里面不能包含空格和冒号。

5、xm的标签可以是中文。

### XML中属性的定义

属性是标记的属性，可以为标记添加附加信息。属性是一个名值对，必须由名称和值组成，属性必须在标记的开始标记或空标记中声明，用”=”为属性指定一个值。

语法如下：

|  |
| --- |
| <标记名称 属性列表/>  <标记名称 属性列表>XXX</标记名称>  <!-- 例如 -->  <桌子 width="40" height='100'/> |

<person></person>

规则：1)一个标签上可以有多个属性。

2)属性名称不能相同。

3)属性名和属性值之间使用“=”连接。

4)多个属性之间使用空格间隔。

4)属性值必须使用引号，单引号和双引号均可使用，如果属性值本身包含双引号，那么有必要使用单引号包围它或者可以使用实体（&quot）引用它。

使用属性的原则：

属性不体现数据的结构，只是数据的附加信息； 一个信息是作为一个标记的属性或子标记，取决于具体问题，不要因为属性的频繁使用破坏XML的数据结构。下面是一个结构清晰的XML文件:

|  |
| --- |
| <楼房 height="23m" width="12m">  <结构>混凝土</结构>  <类别>商用</类别>  </楼房>  下面是一个结构不清晰的XML文件:  <楼房 height="23m" width="12m" 结构="混凝土" 建筑商="华海集团" 类别="商用"></楼房> |

### XML中的注释

格式：<!--注释内容-->

注：注释不可以有嵌套

### XML中的空白被保留

空白被保留是指在XML文档中，空白部分并不会被解析器删除，而是被当作数据一样完整的保留。

### XML中的实体(特殊字符)

如果想要在XML中显示”a<b”不能正常显示，因为把<当做标签来解释。 如果就想要显示，需要对特殊字符<进行转义。使用实体引用来代替“<”字符：

@ ---@amp

< --&lt

>---&gt

“ ---&quot

. --- &apos

### CDATE区:

CDATA是用于需要原文保留的内容，尤其是在解析XML过程中产生歧义的部分，当某个节点的数据有大量需要转义的字符时，那么CDATA就可以发挥其作用。

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。

格式：<![CDATE [内容]]>

例如：

<superType 范围="200" 程度="100%">

<![CDATA[<"溅射"伤害>]]>

</superType>

显示内容：<”溅射”伤害> ，但是CDATA段中不能嵌套另一个CDATA段。

### PI指令（处理指令）

可以再XML中设置样式

<?XML-stylesheet type="text/css" href="1.js"?>

XML：

<? XML version="1.0" encoding="utf=8">

<?XML-stylesheet type="text/css" href="1.js"?>

<person>

<name>1111111111</name>

<age>222222222222</age>

</person>

Js中

name{

background:red;

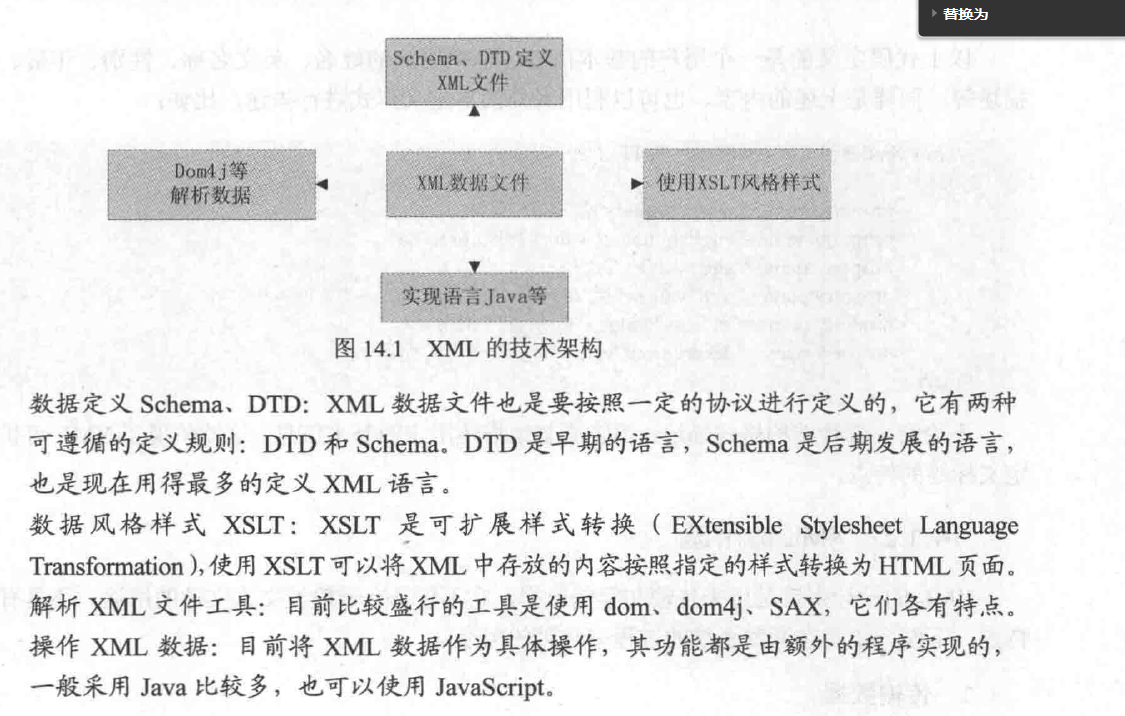
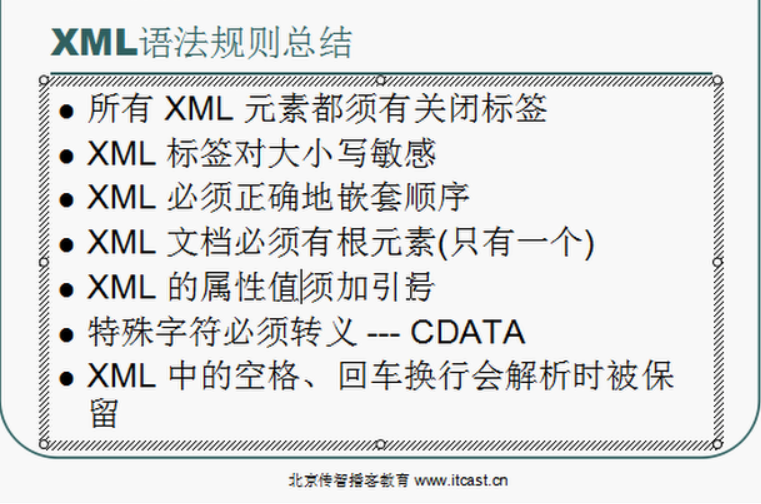
}

age{

background:red;

}

PI设置的样式只能对英文标签起作用，中文标签不起作用



# DTD

## XML的约束

### 为什么需要约束？

DTD(Document Type Definition)文档类型定义：是用来规范XML文件的格式，必须出现在头文件中，一遍XML校验器在一开始便可以得到XML文件的格式定义。DTD是一套关于标记符的语法规则，它定义了可用在文档中的元素、属性和实体，以及这些内容之间的关系。DTD的文后缀名为.dtd

比如现在定义一个person的XML文件，只想要这个文件保存人的信息，比如name， age等，但是，如果在XML文件中写了一个标签<猫>，发现可以正常显示，应为符合语法规范。但是猫肯定不是人的信息，新买来的标签是自定义的，需要技术来规定XML只能出现的元素，这个时候需要约束。

### 验证：

验证方式：格式验证和内容验证

DTD:只验证标签的格式，内容不做验证。(内容的验证一般交给后台java来处理。)

XML的约束技术：dtd的约束和schema约束 （常见的约束）

## dtd的快速入门

创建一个文件后缀名为 .dtd

步骤 ：

1)看XML中有多少个元素，有几个元素，在dtd文件中写几个<!ELEMENT>

2)判断元素是简单元素还是复杂元素。

复杂元素：有子元素的元素。

<!ELEMENT 元素名称 (子元素) >

简单元素：没有子元素

<!ELEMENT 元素名称 (#PCDATE) >//#PCDATE表示一个字符串。

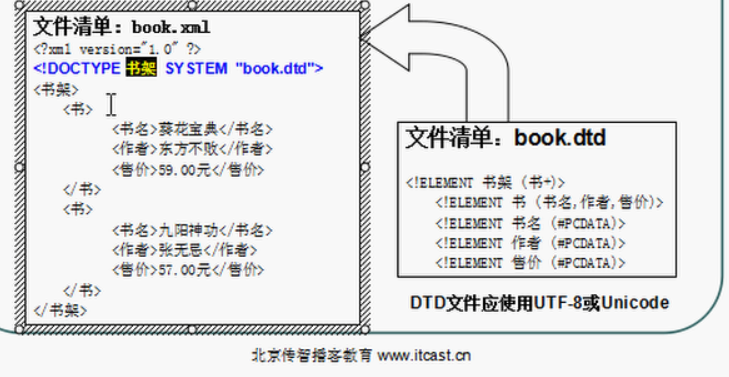
3)在XML文件中引入dtd文件，用来进行约束。

<!DSOCTYPE 根元素名称 SYSTEM “dtd 文件的路径”>

注：只能写在<?xml?>的下边。

例如：

|  |
| --- |
| XML文件  <?xml version="1.0" encoding="utf-8"?>  <!DSOCTYPE person SYSTEM “1.dtd”>  <person>  <name>lee</name>  <age>22</age>  <a>22</a>使用其他标签eclipse会报错。  </person>  DTD文件  <!ELEMENT person (name，age) >复杂元素  <!ELEMENT name (#PCDATA) >简单元素  <!ELEMENT age(#PCDATA) >简单元素 |



打开XML文件使用浏览器打开的，浏览器只负责校验语法，而不负责校验约束。如果校验XML的约束，需要使用工具。可以使用myeclipse进行校验。

## dtd的引入方式（三种）

### 外部引入

外部引入DTD文件：

格式：<!DOCTYPE 根元素名称 SYSTEM “根元素的路径”>

注：

1)System：表示使用当前系统的dtd

### 使用内部的dtd代码(在XML中使用dtd)

<!DOCTYPE 根元素名称{

<!ELEMENT students (student+/？/\*)>

<!—表示students标签中的元素只能是括号中的类型-->

<!ELEMENT student (name,age,score)>

<!—表示student标签中的元素只能是括号中的类型-->

<!ATTLIST student id CDTA(#PCDATE)>

<!—表示student标签中的元素只能是括号中的类型-->

<!ELEMENT name (#PCDATE)>

<!—表示name 为属性，且属性的内容是字符串-->

<!ELEMENT age (#PCDATE)>

<!ELEMENT score (#PCDATE)>

}>

1、+：代表元素出现1-n。

? :代表元素出现0-1.

\*: 代表从元素出现0-n。

### 使用外部的dtd文件（网络上的dtd文件）

格式：<!DOCTYPE 个人元素 PUBLIC “DTD名称” “DTD文档的URL”>

注：

1)PUBLIC 指的是使用网络上的dtd文件。后面要学到框架struct 使用配置文件，使用外部的dtd

## 使用DTD来定义元素

语法:<!ELEMENT 元素名 约束>

### 简单元素：没有子元素的元素

<!ELEMENT name (#PCDATA)>

(#PCDATA) 约束name是字符串类型

EMPTY :元素为空（没有内容）、

ANY：元素可以为空的，也可不为空。

例如:DTD 代码

<!ELEMENT sex EMPTY>

XML代码

<sex><sex>(元素必须为空)

DTD代码<!ELEMENT sex EMPTY>

<sex><sex>

### 复杂元素

格式:<!ELEMENT 元素名称（元素1，元素2，元素3）>

例如:<!ELEMENT 元素名称（school，sex，name）>

注：

1)其中的元素在XML中只能出现一次。如果想要出现多次，可以使用特殊符号

\*：一次或者多次

?：表示元素出现零次或者1次

+：表示元素出现的次数大于1次

<!ELEMENT 元素名称（school+，sex，name）

2)子元素之间用逗号进行隔开,表示元素出现的顺序。

|  |
| --- |
| XML文件  <?xml version="1.0" encoding="utf-8"?>  <!DSOCTYPE person SYSTEM “1.dtd”>  <person>  <name>lee</name>  <age>22</age>  <a>22</a>使用其他标签eclipse会报错。  </person>  DTD文件  <!ELEMENT person (name，age) >复杂元素//元素的出现先后顺序为name，age  <!ELEMENT person (name|age) > //或，表示元素只能出现其中的一个  <!ELEMENT name (#PCDATA) >简单元素  <!ELEMENT age(#PCDATA) >简单元素 |

## 使用dtd来定义属性

### 语法格式：

<!ATTLIST 元素名称 //标签名

属性名 属性类型 约束

属性名 属性类型 约束

......

>

### 属性类型

CDATE:表示属性的取值为普通的文本字符串。

枚举类型：（aa|bb|cc）在一定的范围内出现值，但是只能出现一个。

ID：表示属性的取值不能重复（并且对属性值不能以数字开头）。

例如<!ELEMENT age (#PCDATE)>

<!ATTLIST age ID2 (AA|BB|CC) #REQUIRED> 必须出现的

Id：值只能是字母或者下划线开头”

<!ELEMENT age (#PCDATE)>

<!ATTLIST age ID3 ID #REQUIRED>

<age ID3=”2222”>111</age

### 属性的约束

#REQUIRED:属性必须存在。如果属性不存在，会报错。

#IMPLIED:属性可有可无。

#FIXED: 表示该属性的取值为一个固定值。属性的值只能是设定的值。否则报错。

语法：#FIXED "固定值"。

直接值：表示属性的取值为默认值。

例如：<!ELEMENT age (#PCDATE)>

<!ATTLIST year #IMPLIED “www” >

<age></age> 如果不写属性year时，标签中默认的year属性值为www、如果显式的设置了year的值，那么使用显式的值。

例如：定义多个属性

<!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"

>

### 定义实体

作用：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。

定义实体：<!ENEIEY 实体名称 “实体的值”>

例如：在Dtd中定义<!ENTITY TEST “实体”>

使用实体：&实体名称

在XML中<name>&TEST;</name>

显式为：<name>实体;</name>

注：定义的实体需要写到内部的dtd里面，如果写到外部的dtd里面，某些浏览器得不到\*（指dtd引入方式）

定义引用实体：

概念：在DTD中定义，在XML中使用

语法：<!ENTITY 实体名称 “实体内容”>

引用方式（注意是在XML中使用）：&实体名称;

例如：DTD中定义:

<!ENTITY copyright “版权所有”>

XML中引用:

&copyright;

### 实例

|  |
| --- |
| <span style="font-size:18px;"><?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=“p1">  <姓名>张三</姓名>  <EMAIL>zhang@it315.org</EMAIL>  </联系人>  <联系人 编号=“p2">  <姓名>李四</姓名>  <EMAIL>li@it315.org</EMAIL>  </联系人>  </联系人列表></span> |

案例二

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE TVSCHEDULE [  <!ELEMENT TVSCHEDULE (CHANNEL+)>  <!ELEMENT CHANNEL (BANNER,DAY+)>  <!ELEMENT BANNER (#PCDATA)>  <!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>  <!ELEMENT HOLIDAY (#PCDATA)>  <!ELEMENT DATE (#PCDATA)>  <!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>  <!ELEMENT TIME (#PCDATA)>  <!ELEMENT TITLE (#PCDATA)>  <!ELEMENT DESCRIPTION (#PCDATA)>    <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>  <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>  <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>  <!ATTLIST TITLE RATING CDATA #IMPLIED>  <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>  ]>  <TVSCHEDULE NAME="">  <CHANNEL CHAN="">  <BANNER>CCAV</BANNER>  <DAY>  <DATE>2014-11-17</DATE>  <PROGRAMSLOT>  <TIME>19:00</TIME>  <TITLE>新闻联播</TITLE>  </PROGRAMSLOT>  </DAY>  </CHANNEL>  </TVSCHEDULE> |

# XML解析的简介

## XML解析

### Js使用dom解析标记型文档

根据html的层级结构，在内存中给他分配一个树形结构，将html的标签，属性，文本，都装成对象Document 对象 element对象 属性对象 文本对象 NODE节点对象。

### xml解析

XML是一种通用的数据交换格式,它的平台无关性、语言无关性、系统无关性、给数据集成与交互带来了极大的方便。XML在不同的语言环境中解析方式都是一样的,只不过实现的语法不同而已。

解析方式：DOM解析方式和SAX解析方式

解析对比：

dom方式解析：Document Objext Model官方标准

根据xm1的层级结构在内存中分配一个树形结构，把xm1的标签，属性和文本都封装成对象

缺点：如果文件过大，造成内存溢出

优点：很方便实现增删改操作

sax方式解析：simple API FOR XML

采用事件驱动，边读边解析，从上到下，一行一行的解析，解析到某一个对象，返回对象名称。

缺点：不能实现增删改操作

优点：如果文件过大，不会造成内存溢出，方便实现查询操作。

### 解析器

JAXP：是SUN公司推出的解析标准实现。

JDom:是开源组织推出的解析开发包。

Dom4J:是开源组织推出的解析开发包。开发中最常用的解析方式，是Jdom的升级版，集Jdom的优点。

## Dom解析

### dom解析

　DOM的全称是Document Object Model，也即文档对象模型。在应用程序中，基于DOM的XML分析器将一个XML文档转换成一个对象模型的集合（通常称DOM树），应用程序正是通过对这个对象模型的操作，来实现对XML文档数据的操作。通过DOM接口，应用程序可以在任何时候访问XML文档中的任何一部分数据，因此，这种利用DOM接口的机制也被称作随机访问机制。

　DOM接口提供了一种通过分层对象模型来访问XML文档信息的方式，这些分层对象模型依据XML的文档结构形成了一棵节点树。无论XML文档中所描述的是什么类型的信息，即便是制表数据、项目列表或一个文档，利用DOM所生成的模型都是节点树的形式。也就是说，DOM强制使用树模型来访问XML文档中的信息。由于XML本质上就是一种分层结构，所以这种描述方法是相当有效的。

　DOM树所提供的随机访问方式给应用程序的开发带来了很大的灵活性，它可以任意地控制整个XML文档中的内容。然而，由于DOM分析器把整个XML文档转化成DOM树放在了内存中，因此，当文档比较大或者结构比较复杂时，对内存的需求就比较高。而且，对于结构复杂的树的遍历也是一项耗时的操作。所以，DOM分析器对机器性能的要求比较高，实现效率不十分理想。不过，由于DOM分析器所采用的树结构的思想与XML文档的结构相吻合，同时鉴于随机访问所带来的方便，因此，DOM分析器还是有很广泛的使用价值的。

　优点：

　　　　　　1、形成了树结构，有助于更好的理解、掌握，且代码容易编写。

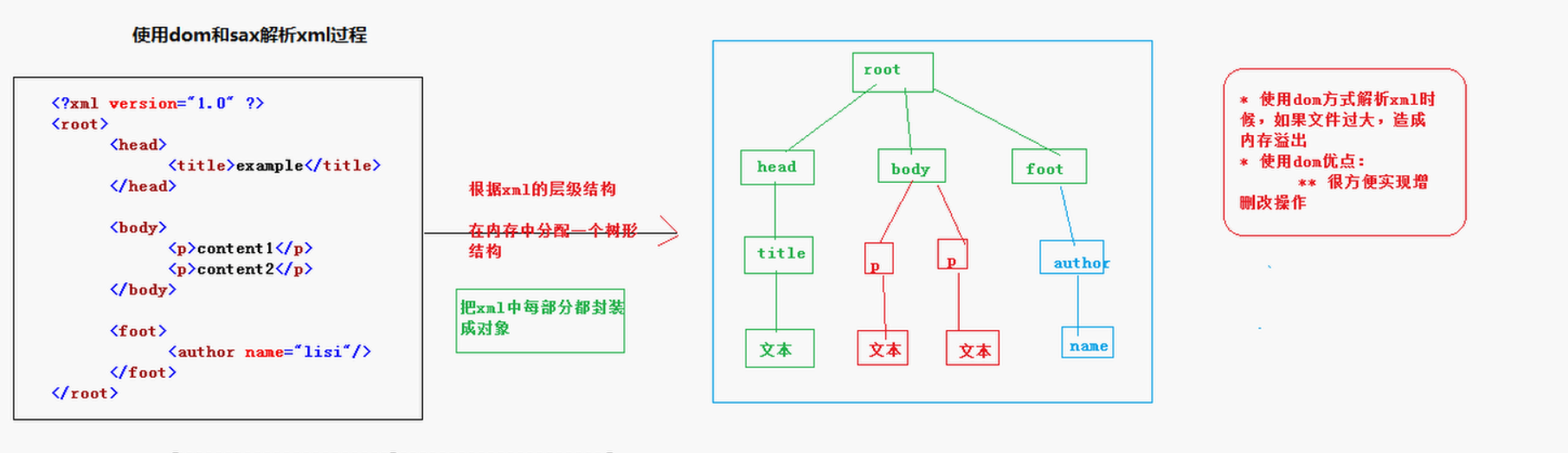
　　　　　　2、解析过程中，树结构保存在内存中，方便修改。容易做增删改操作。

　　　　缺点：

　　　　　　1、由于文件是一次性读取，所以对内存的耗费比较大。

　　　　　　2、如果XML文件比较大，容易影响解析性能且可能会造成内存溢出。

### 原理





### dom解析代码

|  |
| --- |
|  |

### 通用操作：

1、创建解析器工厂

DocumentBuilderFactory builderFactory =

DocumentBuilderFactory.newInstance();

2、根据解析器工厂创建解析器

DocumentBuilder builder = builderFactory.newDocumentBuilder();

3、解析xm1返回document

Document document=builder.parse("XML文件路径");

//这里返回一个Document对象，下面开始对着个对象中的内容进行解析。

# Jaxp解析

## jaxp

Jaxp(Java Api for XML Processing)是javase的一部分。Jaxp解析器在jdk的javax.XML.parsers包里面，四个类：分别是针对dom和sax解析使用的类。

### 针对dom：

DoucmentBuilde:解析器类，这个类是个抽象类不能实例化，此类的实例可以从 DocumentBuilderFactory.newDocumentBuilder() 方法获取到一个方法可以解析XML parse（“XML的路径”）返回是document的整个文档。

在org.w3c.com中的Document是一个接口，父接口是node 在document不到想要的方法，在node中去找在document中的方法。

DocumentBuilderFactory 解析器工厂,也是抽象类。通过newInstance()获取DocumentBuilderFactory的实例

### document中的方法

getElementByTagName（String tagname）这个方法可以得到标签//如果要获得根标签，只获取到一个，其他语言中会可能会有多个。返回集合NodeList

createElement(String tagName) 创建标签

createTextNode(String date) 创建文本

appendChild(Node new Child) 把文本添加到标签下面

removeChild(Node oldChild) 删除节点

getParentNode() 获取到父节点

### NodeList中的方法

遍历NodeList

getlength()得到集合的长度

Item（int index） 下表取到具体的值

for（int i=0；i<list.getlength();i++）

{ list.item(i); }

getTextContent();得到标签里面的内容

### 原始数据。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" standalone="no"?><书架>  <书 ISBN="a11" 出版社="清华大学出版社">  <书名>JavaSE基础</书名>  <作者>张三</作者>  <批发价>35.00元</批发价>  <售价>38.00元</售价>  </书>  <书 ISBN="b11" 出版社="北京大学出版社">  <书名>Android</书名>  <作者>李四</作者>  <售价>38.00元</售价>  </书>  </书架> |

### 获取document对象

|  |
| --- |
| //得到解析工厂DocumentBuilderFactory  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  //得到解析器DocumentBuilder  DocumentBuilder builder = factory.newDocumentBuilder();  //解析指定的XML文档，得到代表内存DOM树的Document对象  Document document = builder.parse("src/book.xml");//xml的路径，注意  注：Document这个接口是w3c包下的。 |

### 得到某个具体节点内容：打印第2本书的作者

|  |
| --- |
| public static void test1(Document document){  //根据标签的名称获取所有的作者元素  NodeList nodeList = document.getElementsByTagName("作者");  //按照索引取第2个作者元素  Node node = nodeList.item(1);  //打印该元素的文本  String text = node.getTextContent();通过该方法获取该节点的文本。  System.out.println(text);  } |

### 遍历所有元素节点:打印元素的名称

|  |
| --- |
| public static void test2(Node node){  //判断当前节点是不是一个元素节点  if(node.getNodeType() == Node.ELEMENT\_NODE){  //如果是：打印他的名称  System.out.println(node.getNodeName());  }  //查找子节点  NodeList nodeList = node.getChildNodes();  int len = nodeList.getLength();//获取节点的个数  for (int i = 0; i < len; i++) {  Node n = nodeList.item(i);//item获取节点  test2(n);  }  } |

### 修改某个元素节点的主体内容:把第一本书的售价改为38.00元

|  |
| --- |
| public static void test3(Document document) throws Exception{  //找到第一本书的售价  NodeList nodeList = document.getElementsByTagName("售价");  //设置其主体内容  Node node = nodeList.item(0);  node.setTextContent("38.00元");  //把内存中Document树写回xml文件中，  TransformerFactory factory = TransformerFactory.newInstance();  Transformer ts = factory.newTransformer();  ts.transform(new DOMSource(document), new StreamResult("src/book.xml"));  } |

### 向指定元素节点中增加子元素节点:第一本中增加子元素<内部价>29.00</内部价>

|  |
| --- |
| public static void test4(Document document) throws Exception{  //创建一个新的元素并设置其主体内容  Element e = document.createElement("内部价");  e.setTextContent("29.00元");  //找到第一本书元素  Node firstBookNode = document.getElementsByTagName("书").item(0);  //把新节点挂接到第一本书上  firstBookNode.appendChild(e);  //把内存中Document树写回XML文件中  TransformerFactory factory = TransformerFactory.newInstance();  Transformer ts = factory.newTransformer();  ts.transform(new DOMSource(document), new StreamResult("src/book.xml"));  } |

### 向指定元素节点上增加同级元素节点:在第一本书的售价前面增加批发价

|  |
| --- |
| public static void test5(Document document) throws Exception{  //创建一个新的元素并设置其中的主题内容  Element e = document.createElement("批发价");  e.setTextContent("35.00元");  //找到第一本书的售价  Node firstPrice = document.getElementsByTagName("售价").item(0);  //在售价的前面加入新建的元素：增加子元素一定要用父元素来增加  firstPrice.getParentNode().insertBefore(e, firstPrice);  //把内存中Document树写回XML文件中  TransformerFactory factory = TransformerFactory.newInstance();  Transformer ts = factory.newTransformer();  ts.transform(new DOMSource(document), new StreamResult("src/book.xml"));  } |

### 删除指定元素节点：删除内部价

|  |
| --- |
| public static void test6(Document document) throws Exception{  //找到内部价节点，用父节点删除  Node n = document.getElementsByTagName("内部价").item(0);  n.getParentNode().removeChild(n);  //把内存中Document书写回XML文件中  TransformerFactory factory = TransformerFactory.newInstance();  Transformer ts = factory.newTransformer();  ts.transform(new DOMSource(document), new StreamResult("src/book.xml"));  } |

### 操作XML文件属性:打印第一本书的出版社

|  |
| --- |
| public static void test7(Document document){  //得到第一本书  Node n = document.getElementsByTagName("书").item(0);  //打印指定属性的取值  Element e = (Element)n;  System.out.println(e.getAttribute("出版社"));  } |

### 添加一个出版社属性给第二本书

|  |
| --- |
| public static void test8(Document document) throws Exception{  //得到第二本书  Node n = document.getElementsByTagName("书").item(1);  //打印指定属性的取值  Element e = (Element)n;  e.setAttribute("出版社", "北京大学出版社");  //把内存中Document树写回XML文件中  TransformerFactory factory = TransformerFactory.newInstance();  Transformer ts = factory.newTransformer();  ts.transform(new DOMSource(document), new StreamResult("src/book.xml"));  } |

Node p1 = list.item(0);

Element sex1=document.createElement("sex");

Text text1 = document.createTextNode("nv");

sex1.appendChild(text1);

//回写写操作

TransformerFactory transformerFactory = TransformerFactory.*newInstance*();

Transformer transformer = transformerFactory.newTransformer();

transformer.transform(**new** DOMSource(document),**new** StreamResult("src/1.XML"));

}

# Schema

## schema

XML Schema 本身也是一种XML构造，它用来描述[哪个元素，在什么时候出现]，[该元素具有什么样的属性]等等，也就是说，XML Schema是对XML的树形构造加以描述说明的一种语言。原本，使用DTD对XML的树形构造加以描述说明，但DTD存在严重的局限性，DTD不能定义数据的类型，语法也与XML语言完全不一样，在使用的便利性，数据结构表达的严谨性上存在问题。

Schema的约束 （schema是一个约束文件 xml是被约束文件）XML Schema文件自身就是一个XML文件，但它的扩展名通常为.xsd和XmL文件一样，一个XMLSchema文档也必须有一个根结点，但这个根结点的名称为Schema。

Dtd的语法<!Element 元素名称 约束>

特点

1、Schema 符合xml的语法，xml语句。

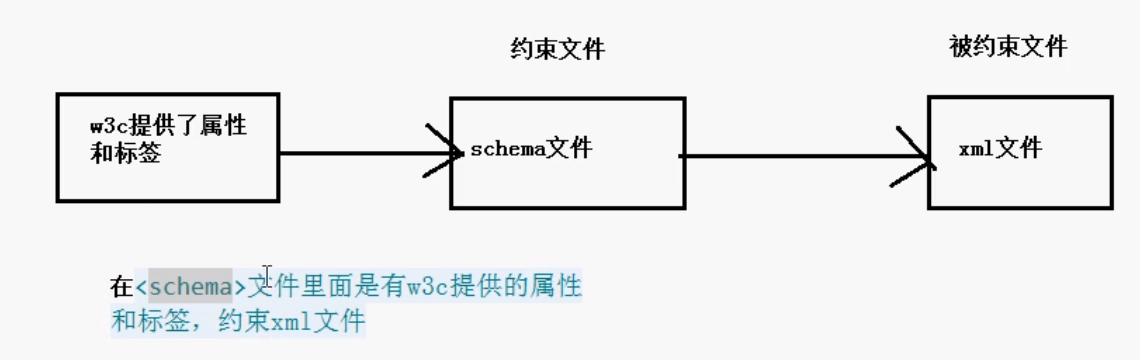
2、一个xml中可以有多个schema 多个schema使用名称空间区分(类似于java中的包名)

3、Dtd开面的PCDATE,但是在schema里面可以支持更多的数据类型

例如：年龄只能是整数，在schenma里面可以直接定义一个整数

4、Sschema出现的原因是替代dtd，但是Sschema结构和语法比较复杂，并没有完全替代。

## schema原理



## schema的快速入门

1）Xml Schema也是一个xml文件，其扩展名通常为.xsd。

2）Xml Schema文档必须有一个根节点，并且这个根节点固定为<Schema>

3）一个Xml Schema，通常称为模式文档（约束文档），遵循这个文档写的xml称之为实例文档。

4）编写一个约束文档后，通常需要把则个文件中声明的元素绑定到一个URI地址上，即定义为一个命名空间，以后xml文件就可以通过这个URI（命名空间）来告诉解析引擎，xml文档中编写的元素来自哪里，被谁约束。

### 创建schema文件

1、创建schema文件,后缀名是.xsd

W3c规定使用schema作为根节点

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <schema xmlns=*"http://www.w3.org/2001/XMLSchema"*  targetNamespace=*"http://www.example.org/1"*  xmlns:tns=*"http://www.example.org/1"*  elementFormDefault=*"qualified"*>  </schema> |

A、xmlns(XML Namespace)：引入命名空间（namespace）：默认是W3C的。这个默认引用， 只能出现一次。非默认引用需要加别名：如 xmlns:tns="http://www.example.org/NewXMLSchema" ，增加了别名 tns 。一个文档可以引入多个命名空间。

B、targetNamespace：定义命名空间，其实Xml Schema的本质就是定义一个命名空间，共其他Xml引用。这边的命名空间定义为URI为 "http://www.example.org/NewXMLSchema" ，引用的时候也必须是这个。

C、elementFormDefault：控制元素。有两个取值："qualified"-定义的所有元素都绑定为这个命名空间； unqualified:表示只有根元素绑定为这个命名空间。

D、attributeFormDefault：控制属性的。用法与elementFormDefault一样。

### 步骤

|  |
| --- |
| <person>  <name>sx</name>  <age>20</age>  </person> |

1)看xml中有几个元素，有几个元素就写几个Element

|  |
| --- |
| <element name=*"pserson"*></element>  <element name=*"name"* type=*"string"*></element>//名字和类型  <element name=*"age"* type=*"int"*></element>//名字和类型 |

2)看简单的元素和复杂的元素

如果是复杂元素

|  |
| --- |
| <element name=*"person"*>  <complexType>  <sequence>  子元素  </sequence>  </complexType>  </element> |

3)简单元素

|  |
| --- |
| <element name=*"person"*>  <complexType>  <sequence>//子元素要写在sequence里面  <element name=*"name"* type=*"string"*></element>  <element name=*"age"* type=*"int"*></element>  </sequence>  </complexType>  </element> |

4)在被约束文件中引入约束的文件

|  |
| --- |
| <person xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://www.example.org/1"*  xsi:schemaLocation=*"http://www.example.org/1 1.xsd"*><!--xis起别名 -->跟标签的位置 |

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance：表示是一个被约束的文件*

xmlns=*"http://www.example.org/1" 约束文档里的targetNameSpace*

xsi:schemaLocation=*"http://www.example.org/1 1.xsd*

*targetNameSpace 空格 约束文档的路径*

## 命名空间（nameSpace）

xmlns：XML Namespace（xml名称空间）名称空间是W3C推荐标准提供的一种统一命名XML文档中的元素和属性的机制。使用名称空间可以明确标识和组合XML文档中来自不同标记词汇表的元素和属性，避免了名称之间的冲突。使用过DTD的人应该知道，命名冲突是DTD的一大问题，而Schema里引入了命名空间的概念，就很好的解决了这个问题。

### 命名冲突

|  |
| --- |
| 在 XML 中，元素名称是由开发者定义的，当两个不同的文档使用相同的元素名时，就会发生命名冲突。这个 XML 文档携带着某个表格中的信息：  <table>  <tr>  <td>Apples</td>  <td>Bananas</td>  </tr>  </table>  这个 XML 文档携带有关桌子的信息（一件家具）  <table>  <name>African Coffee Table</name>  <width>80</width>  <length>120</length>  </table>  假如这两个 XML 文档被一起使用，由于两个文档都包含带有不同内容和定义的 <table> 元素，就会发生命名冲突。XML 解析器无法确定如何处理这类冲突。 |

解决方法：使用前缀来避免命名冲突

|  |
| --- |
| 此文档带有某个表格中的信息：  <h:table>  <h:tr>  <h:td>Apples</h:td>  <h:td>Bananas</h:td>  </h:tr>  </h:table>  此 XML 文档携带着有关一件家具的信息：  <f:table>  <f:name>African Coffee Table</f:name>  <f:width>80</f:width>  <f:length>120</f:length>  </f:table>  现在，命名冲突不存在了，这是由于两个文档都使用了不同的名称来命名它们的 <table> 元素 (<h:table> 和 <f:table>)。通过使用前缀，我们创建了两种不同类型的 <table> 元素。 |

### 使用命名空间

|  |
| --- |
| 这个 XML 文档携带着某个表格中的信息：  <h:table xmlns:h="http://www.w3.org/TR/html4/">  <h:tr>  <h:td>Apples</h:td>  <h:td>Bananas</h:td>  </h:tr>  </h:table>  此 XML 文档携带着有关一件家具的信息  <f:table xmlns:f="http://www.w3school.com.cn/furniture">  <f:name>African Coffee Table</f:name>  <f:width>80</f:width>  <f:length>120</f:length>  </f:table>  与仅仅使用前缀不同，我们为 <table> 标签添加了一个 xmlns 属性，这样就为前缀赋予了一个与某个命名空间相关联的限定名称。 |

2、 简单元素写在复杂元素中

<element name="person">

<complexType >

<sequence>

<element name="name" type="string"></element>

<element name="age" type="int"></element>子元素

</sequence>

</complexType>

</element>

<sequence> 指示器 sequence 表示元素的出现按照顺序来写，将xml中的元素调换位置会报错

<all>表示元素只能出现一次

<choice>元素只能出现其中的一个

maxOccurs

<element name="name" type="string" maxOccurs=“unbounded”></element>

表示改元素可以出现的次数 表示出现无限次数

<any><any> :表示任意元素

Attribute可以约束属性

1、 写在复杂的元素里面

2、 写在</complexType>之前

<attribute name="id1" type="int" use="required"></attribute>

name：属性名称

type：属性类型

use:属性是否必须出现 required

3、 在xml中引入约束的文件

<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" //与xsd相同便是该xml文件是是一个被约束文件

xmlns=http://www.example.org/1 //约束文档的地址或者路径

xsi:schemaLocation="http://www.example.org/1 1.xsd">

// targetNamespace 空格 约束文档的路径地址

<name>zhangsan </name>

<age>20</age>

person>

xsd文件

<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.example.org/1" xmlns:tns="http://www.example.org/1" elementFormDefault="qualified">

<element name="person">

<complexType >

<sequence>

<element name="name" type="string"></element>

<element name="age" type="int"></element>

</sequence>

</complexType>

</element>

</schema>

复杂的约束

通过 别名：属性 对属性进行限制

别名的名字随便，但是别名的名字不能相同

Xml使用多个schema

如果schema部门名称的元素为任何类型 引入多个schema，通过别名来引入

Xml中的代码，实现对xml中代码的约束 不同的约束引入不同的别名来区分

e <employee age="30">

<!--部门名称-->I

<dept：name>100</dept:name>引入dept中的限制

<！--员工名称-->

<name>王晓晓</name>

</employee>

</company>

## 3、sax的解析原理

解析xml的两种技术 dom和sax

Dom解析根据xml的层级结构在内存中分配已个树形结构

把xml中的标签属性，文本封装成对象

Sax方式:事件驱动，边读变解析

（api文档）在java.xmlparsers包里面 有两个类

抽象类SAXParser

得到实例 从SAXParseFactory.newSAXParser（）方法获得

Parse（File F,DefaultHandler dh）

两个参数

第一个参数 xml的路径

事件处理器

SAXParserFactory

得到实例 newInstance（）方法得到

Sax的执行过程

当解析到开始标签是自动执行startElement方法

当解析到文本的时候，自动执行characters方法

当解析到结束标签时自动执行endElement 方法

4、使用jasp的dax方式解析xml

Sax方式不能实现增删改操作，只能执行查询操作

打印出整个文档

执行parse方法第一个参数xml路径，第二个参数是时间处理器，创建一个雷，继承事件处理器的类，重写里面的三个方法

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

SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();

SAXParser saxParser = saxParserFactory.newSAXParser();

saxParser.parse("src/1.xml",new MyDefault());

}

}

class MyDefault extends DefaultHandler {

@Override

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

// TODO 自动生成的方法存根

super.startElement(uri, localName, qName, attributes);

System.out.print(qName);

}

@Override

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

// TODO 自动生成的方法存根

super.endElement(uri, localName, qName);

System.out.print(qName);

}

@Override

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

// TODO 自动生成的方法存根

super.characters(ch, start, length);

System.out.print(new String(ch, start, length));

}

获取到所有name元素的值

class MyDefault1 extends DefaultHandler{

boolean flag = false;

@Override

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

// TODO 自动生成的方法存根

super.startElement(uri, localName, qName, attributes);

if("name" ==qName) {

flag = true;

}

}

@Override

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

// TODO 自动生成的方法存根

super.characters(ch, start, length);

if(flag==true) {

System.out.println(new String(ch, start, length));

}

}

@Override

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

// TODO 自动生成的方法存根

super.endElement(uri, localName, qName);

if("name" ==qName) {

flag = false;

}

}

}

只打印name值

class MyDefault1 extends DefaultHandler{

boolean flag = false;

int index = 1;

@Override

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

// TODO 自动生成的方法存根

super.startElement(uri, localName, qName, attributes);

if("name" ==qName) {

flag = true;

}

}

@Override

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

// TODO 自动生成的方法存根

super.characters(ch, start, length);

if(flag==true && index == 2) {

System.out.println(new String(ch, start, length));

}

}

@Override

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

// TODO 自动生成的方法存根

super.endElement(uri, localName, qName);

if("name" ==qName) {

flag = false;

index++;

}

}

}

\*\*\*执行parse方法，第一个参数xm1路径，第二个参数是事件处理器

\*\*\*创建一个类，继承事件处理器的类，

\*\*\*重写里面的三个方法

\*获取到所有的name元素的值

\*\*定义一个成员变量flag=false

\*\*判断开始方法是否是name元素，如果是name元素，把flag值设置成true

\*\*如果flag值是true，在characters方法里面打印内容

\*\*当执行到结束方法时候，把flag值设置成false

\*获取第一个name元素的值I

\*\*定义一个成员变量idx=1

\*\*在结束方法时候，idx+1 idx+H

\*\*想要打印出第一个name元素的值，

-在characters方法里面判断，

--判断flag=true并且idx==1，在打印内容

## 5、使用dom4j解析xml

DOM在解析的时候把整个XML文件映射到Document的树型结构中，XML中的元素、属性、文本都能在Document中看清，但是它消耗内存、查询速度慢。SAX是基于事件的解析，解析器在读取XML时根据读取的数据产生相应的事件，由应用程序实现相应的事件处理，所以它的解析速度快，内存占用少。但是它需要应用程序自身处理解析器的状态，实现起来比较麻烦，而且它只支持对XML文件的读取，不支持写入。

Dom4j 是一个组织，针对xml得劲解析，提供解析器dom4j

Dom4j不是javase的一部分，要想使用第一步需要怎么

### 导入dom4j提供的jar包(项目上有红色感叹号，表示缺少jar包。)

导入方法：1)创建一个lib文件夹，复制文件到lib文件夹下

2)右键点击jar包 build path add to buildpath

3)看到jar包 变成奶瓶的样子表示导入成功。

4)在dom4j中，解析的xml文件路径必须为全部英文。

### 重点掌握：

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

//0.引入jar包

//1.读取文件形成DOM树

SAXReader reader = new SAXReader();

Document doc = reader.read(new File("students.xml"));

//2.获取根节点students

Element studentsElem = doc.getRootElement();

//3.从根节点students中获取所有的student信息

Iterator<Element> it = studentsElem.elementIterator();

//4.处理所有的student信息

while(it.hasNext()){

//取出一个学生信息

Element stuElem = it.next();

//获取一个学生的所有属性并输出：id

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

for(Attribute attr:attributes){

String aname = attr.getName();

String atext = attr.getText();

System.out.println(aname+"-----"+atext);

}

//获取一个学生的所有子元素并输出：name age score

Iterator<Element> it2 = stuElem.elementIterator();

while(it2.hasNext()){

//获取一个子元素 name or age or score

Element nasElem = it2.next();

//获取一个子元素名称： name age score

String ename = nasElem.getName();

//获取一个子元素的文本：李明 23 89

String etext = nasElem.getText();

//输出

System.out.println(ename+"------>"+etext);

}

System.out.println();

}

}

### 得到document

SAXReader reader = new SAXReader();

Document document = reader.read(url);

document的父接口是Node 在document中找不到方法在Node中去找node 中的方法 。

document getRootElement()获取根节点返回的是Element, Element 也是一个接口，父接口是Node。

Element和Node中的方法

getParent（）获取父节点

addElement（）添加标签

实现方法 element(qname)表示获取标签下面的第一个字标签

Qname：标签 的名称

elements(qname)

获取下级字标签

qname:标签名称

elements():获取下面所有级别的字标签

6、使用dom4j查询dom

<p1>

<name>zhangsan</name>

<age>20</age>

</p1>

<p1>

<name>lisi</name>

<age>30</age>

</p1>

需求：查询name元素里面的值

1、 创建解析器

2、 得到document

3、 得到根节点

4、 得到所有的p1的标签

Element（“p1” ）返回list集合

遍历list的每一个p1

5、 得到name

6、 得到name中的值

7、 public static void selectName() throws Exception{

8、 SAXReader saxReader = new SAXReader();

9、 Document document = saxReader.read("src/1.xml");

10、 Element root = document.getRootElement();

11、 List<Element> list=root.elements("p1");

12、 for (Element element : list) {

13、 Element name1=element.element("name");

14、 String s = name1.getText();

15、 System.out.println(s);

16、

17、 }

得到第一个name的值

public static void selectName1() throws Exception{

SAXReader saxReader = new SAXReader();

Document document = saxReader.read("src/1.xml");

Element root = document.getRootElement();

Element p1 = root.element("p1");

Element name1 = p1.element("name");

String s = name1.getText();

System.out.println(s);

}

得到第二个name

\*1、创建解析器

\*2、得到document

\*3、得到根节点

\*4、得到所有的p1

\*\*返回list集合

\*5、遍历得到第二个p1I

\*\*使用list下标得到get方法，集合的下标从0开始，想要得到第二个值，下标写1

\*6、得到第二个p1下面的name

\*7、得到name的值

public static void selectName2() throws Exception{

SAXReader saxReader = new SAXReader();

Document document = saxReader.read("src/1.xml");

Element root = document.getRootElement();

List<Element> list = root.elements("p1");

Element p2=list.get(1);

Element name2=p2.element("name");

String s = name2.getText();

System.out.println(s);

}

7、使用dom4j实现添加操作

<p1>

<name>zhangsan</name>

<age>20</age>

</p1>

<p1>

<name>lisi</name>

<age>30</age>

</p1>

（1） 操作在第一个p1标签的末尾添加一个标签<sex></sex>

（2） \*1、创建解析器

（3） \*2、得到document

（4） \*3、得到根节点

（5） \*4、获取到第一个p1

（6） \*使用element方法

（7） \*5、在p1下面添加元素

（8） \*在p1上面直接使用addElement（\"标签名称\"）方法返回一个Element

（9） \*6、在添加完成之后的元素下面添加文本

（10） \*在sex上直接使用setText（\"文本内容\"）方法

（11） \*7、回写xml

（12） \*格式化outputFormat，使用createprettyPrint方法，表示一个漂亮的格式

（13） \*使用类xMLWriter直接new这个类，传递两个参数

（14） I\*\*\*第一个参数是xm1文件路径 new Fileoutputstream(\"路径\"）

（15） \*\*\*第二个参数是格式化类的值

（16） \*\*/

public static void addSex() throws Exception {

SAXReader saxReader = new SAXReader();

Document document = saxReader.read("src/1.mxl");

Element root = document.getRootElement();

Element p1 = root.element("p1");

Element sex1 = p1.addElement("sex");

sex1.setText("nv");

OutputFormat format = OutputFormat.createCompactFormat();

XMLWriter xmlWriter = new XMLWriter(new FileOutputStream("src/1.xml"),format);

xmlWriter.write(document);

xmlWriter.close();

（1） }

8、在特定的位置添加一盒元素

实现 在第一个p1下面的age标签之前调价<school>edit</school>

\*1、创建解析器

\*2、得到document

\*3、得到根节点

\*4、获取到第一个p1

\*5、获取p1下面的所有的元素

\*\*elements（）方法返回list集合

\*\*使用list里面的方法，在特定位置添加元素

\*\*首先创建元素在元素下面创建文本

-使用DocumentHelper类方法createElement创建标签

-把文本添加到标签下面使用setText（\"文本内容”）方法

\*\*\*list集合里面的 add(int index,E element)

-第一个参数是位置下标，从0开始

-第二个参数是要添加的元素

\*6、回写xm1

public static void addAgeBrfore() throws Exception{

SAXReader saxReader = new SAXReader();

Document document = saxReader.read("src/1.mxl");

Element root = document.getRootElement();

Element p1 = root.element("p1");

List list = p1.elements();

Object element;

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

school.setText("ecit");

list.add(1, school);

OutputFormat format = OutputFormat.createPrettyPrint();

XMLWriter xmlWriter = new XMLWriter(new FileOutputStream("src/1.xml"),format);

xmlWriter.write(document);

xmlWriter.close();

}

封装

public static Document getDocument(String path) {

SAXReader reader = new SAXReader();

try {

Document document =reader.read(path);

return document;

} catch (DocumentException e) {

// TODO 自动生成的 catch 块

e.printStackTrace();

}

return null;

}

public static void xmlWriters(String path,Document document) {

try {

OutputFormat format = OutputFormat.createPrettyPrint();

XMLWriter xmlWriter = new XMLWriter(new FileOutputStream(path),format);

xmlWriter.write(document);

xmlWriter.close();

}catch(Exception e) {

e.printStackTrace();

}

9、使用dom4j实现修改节点的操作

\*1、得到document

\*2、得到根节点，然后再得到第一个p1元素

\*3、得到第一个p1下面的age

element(\"\")方法

\*4、修改值是30

\*\*使用setText（\"文本内容\")方法

\*5、回写xml

\*

\*\*/I

public static void modifyAge() throws Exception {

Document document = Dom4jUtils.getDocument("src/a.xml");

Element root =document.getRootElement();

Element p1 = root.element("p1");

Element age = p1.element("age");

age.setText("300");

Dom4jUtils. xmlWriters("src/a.xml", document);

}

快速导包 ctrl shift o

10、使用dom4j实现删除节点的操左

删除第一个p1下面的<school>ecit</school> 的元素

<person>

<p1>

<name>zhangsan</name>

<school>>ecit</school>

<age>20</age>

</p1>

<p1>

<name>lisi</name>

<age>30</age>

</p1>

</person>

\*1、得到document

\*2、得到根节点

\*3、得到第一个p1标签

\*4、得到第一个p1下面的school元素

\*5、删除（使用p1删除school)

\*\*得到school的父节点

-第一种直接得到p1

-使用方法getparent方法得到

\*删除操作

-在p1上面执行remove方法删除节点

\*6、回写xml

public static void delSch() throws Exception{

Document document = Dom4jUtils.getDocument("src/a.xml");

Element root =document.getRootElement();

Element p1 = root.element("p1");

Element sch = p1.element("school");

p1.remove(sch);

Dom4jUtils. xmlWriters("src/a.xml", document);

}

10、使用dom4j获取属性的操作

获取第一标签id1的值

\*1、得到document

\*2、得到根节点

\*3、得到第一个p1元素

\*4、得到p1里面的属性值

-pl.attributeValue(\"id1\");

-在p1上面执行这个方法，里面的参数是属性名称

public static void getValue() throws Exception {

Document document = Dom4jUtils.getDocument("src/a.xml");

Element root =document.getRootElement();

Element p1 = root.element("p1");

String s = p1.attributeValue("id1");

System.out.println(s);

}

12使用dom4j指出xpath的操作（xpath）

可以直接获取某个元素

第一种形式 /AAA/BBB/CCC 表示一层一层的AAA下面DDD下面BBB

第二种形式 //BBB 表示这个名称相同的 表示只要是名称是BBB都得到

//@id id属性相统统都可以得到

第三种形式 /\*表示元素

第四种形式 /AAA/BBB[1] AAA中的第一个BBB元素

/AAA/BBB[last()] AAA中最后一个BBB元素

第五种形式 //BBB[@id]表示只要BBB有id属性都可以得到

第六种形式 //BBB[@id=“aaa”] 表示只要BBB有id属性并且id的属性为“aaa”都可以得到（三个条件）

13、使用dom4j支持xpath具体操作

默认情况下 dom4j里面有xpath

第一步需要 引入指出xpath的jar包，使用jaxen-1.1-beat-6.jar

需要吧jar包导入到项目中

在dom4j里面提供了两个方法，用来支持xpath

selectNodes（“xpath表达式”）

获取多个节点

selectSingleNode（““xpath表达式”）

获取单一的节点

使用xpath查询xml中所有name元素的值

所有name元素 xpath表示//name

public static void test1() throws Exception{

Document document = Dom4jUtils.getDocument("src/a.xml");

List<Node> list = document.selectNodes("//name");

for ( Node node : list) {

String s = node.getText();

System.out.println(s);

}

}

使用xpath实现：获取第一个p1下面的值

//p1[@id1]=’aaa’]/name

使用到selectSingleNode（“”//p1[@id1=’aaa’]/name“）单引号

\*1、得到documentI

\*2、直接使用selectsingleNode方法实现

\*-xpath：//p1[@idl='aaaa']/name

”）

public static void test2() throws Exception{

Document document = Dom4jUtils.getDocument("src/a.xml");

Node name1 = document.selectSingleNode("//p1[@id1='aaaa']/name");

String s1 = name1.getText();

System.out.println(s1);

}

14、实现简单的学生管理系统