**XML概述**

## 一、xml基础详解

**1、概述**

xml:即可扩展标记语言，xml是互联网数据传输的重要工具，它可以跨越互联网任何的平台，不受编程语言和操作系统的限制，可以说它是一个拥有互联网最高级别通行证的数据携带者。xml是当前处理结构化文档信息中相当给力的技术，xml有助于在服务器之间穿梭结构化数据，这使得开发人员更加得心应手的控制数据的存储和传输。Xml用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。Xml是标准通用标记语言（SGML）的子集，非常适合Web传输。XML提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。

**2、xml的特点及作用**

(1)特点

v xml与操作系统、编程语言的开发平台都无关；

v 实现不同系统之间的数据交互。

(2)作用

v 配置应用程序和网站；

v 数据交互；

v Ajax基石。

在配置文件里边所有的配置文件都是以XMl的格式来编写的。

跨平台进行数据交互，它可以跨操作系统，也可以跨编程语言的平台。

Ajax是现在目前比较流行的一个网络交互的技术。Ajax里面最后一个x实际上就是xml的缩写。Xml（Extensible Markup Language）是可扩展标记语言

一个xml文档必须要有第一行的声明和它的文档元素的描述信息就可以了。

**3、xml声明**

注：xml声明一般是xml文档的第一行；xml声明由以下几个部分组成：

**4、根元素**

它是xml文档里面唯一的；它的开始是放在最前面，结束是放在最后面。

**5、元素**

(1) 所有的xml元素都必须有结束标签；

(2) xml标签对大小写敏感；

(3) xml必须正确地嵌套；

(4) 元素的命名规则：

名称中可以包含字母、数字或者其他的字符；

名称不能以数字或者标点符号开始；

名称中不能包含空格。

(5) 空元素

**6、属性**

（1）语法

<元素名 属性名=“属性值”/>

例：<Student ID=“S100”>

<Name>Tom</Name>

</Student>

(2)注意：

属性值用双引号包裹；一个元素可以有多个属性，它的基本格式为：

<元素名 属性名=“属性值” 属性名=“属性值”>;

属性值中不能够直接包含<.”,&。

**7、实体：**

在xml中，一些字符拥有特殊的意义。如果把字符“<”放在xml元素中，会发生错误，这是因为解析器会把它当作新元素的开始，这样会产生xml错误：

为了避免这个错误，请用实体引用来代替“<”字符：

xml中5个预定义实体

**8、注释**

注:注释内容中不要出现”--”;不要把注释放在标签中间；注释不能嵌套。

**9、总结：**

（1）xml描述的是文档的内容与语义，而不是文档应当如何显示；

（2）格式正规（well formed）的xml文档

遵循如下规则的xml文档称为格式正规的xml文档：

v 必须有xml声明语句；

v 必须有且仅有一个根元素；

v 标签大小写敏感；

v 属性值用双引号；

v 标签成对；

v 空标签关闭；

v 元素正确嵌套。

(3)有效的（valid）xml文档。首先xml文档是个格式正规的xml文档，然后又需要满足DTD的要求，这样的xml文档称为有效的xml文档；

**10、xml命名空间（xml Namespaces）**

（1）xml命名空间提供避免元素命名冲突的方法。

（2）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。

**11、xml命名空间示例**

（1）使用前缀示例，与仅仅使用前缀不同，我们为标签添加了一个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。

## 二、DTD技术——xml文件的验证机制

**1、DTD概述：**

文档类型定义——Document Type Definition

DTD用来描述xml文档的结构，一个DTD文档包含：

元素的定义规则；元素之间的关系规则；属性的定义规则。

**2、为什么要用DTD**

（1）有了DTD，每个xml文件可以携带一个自身格式的描述,所谓格式描述就是我的xml文档里面可以写哪些东西，比如元素、属性；

（2）有了DTD，不同组织的人可以使用一个通用DTD用来交换数据，xml是随意定义的，有了DTD，我们可以限制多个公司之间按照这种DTD的规则来编写xml文档，

由于DTD都是统一格式，所以不同的公司之间，不同的组织之间就可以用这种通用的xml文档格式进行交互数据了；

（3）应用程序可以使用一个标准DTD校检从外部接收的xml数据是否有效。

**3、如何编写一个DTD**

DTD分成三大类，第一是内部DTD,第二是外部DTD,第三是内外结合的DTD。

（1）内部DTD文档(Students.xml)：

（2）就是说这个xml文档里面既包含xml定义，又包含DTD的定义。

这个[]里面是DTD的定义的内容，就是第一个叫内部DTD文档。

（3）外部DTD文档（Students.xml Students.dtd）:

所谓外部就是DTD的定义和我们的xml文档是在不同的文件里面。

那么外部的DTD如何使用呢？那就需要在前面的xml文档的根元素的上面这个位置跟刚才内部的是一样的，要写上

这类年改革文档在同一个文件夹下面，就可以了。

有内部DTD了，为什么还要学习外部DTD呢？？

内部的DTD,它只能验证当前的这个xml文档。那假如说，我们有一个DTD需要验证多个xml文档的话，那就直接使用外部DTD就行。

（4）内外部DTD文档结合

**4、DTD的具体定义——元素的定义**

元素名称：是指xml文档里面的元素名称

元素类型：

（1）EMPTY

示例第一个是对的，第二个是错误的，因为它有内容了。

（2）#PCDATA

第一个是对的，第二个是错误的。

（3）纯元素类型的DTD语法

（4）ANY

**5、定义属性的语法结构**

xml文档携带数据的方式，不仅仅是xml的元素的内容，还可以是元素的属性。

所以说在DTD里面我们有必要对属性进行限制。

（1）编写属性验证的语法规则：

（2）属性类型——CDATA

CDATA是表示属性的值，可以是任何字符，这里包括数字和中文。

问题一：属性的CDATA与元素的CDATA节有何区别呢？？

属性的CDATA表示属性值是任何的字符，元素里的CDATA节是表示在CDATA节里面的内容不被xml文档进行解析。

问题二：#PCDATA与CDATA有何区别？？

#PCDATA是限制元素里面的内容是字符类型的，而这个CDATA是限制属性里面的内容是字符类型。

（3）属性类型——ID

表明该属性的取值必须是唯一的。

（4）属性类型——IDREF/IDREFS

v IDREF是指它的值要指向文档当中其它地方声明的ID。把IDREF当成数据库的外键，把ID当成主键理解就行；

v IDREFS同IDREF，但是可以具有由空格分开的多个引用；

（5）属性类型——Enumerated

预先定义了一些值，属性的值必须在所列出的值的范围内

（6）属性特点——#REQUIRED

它表示元素的所有的实例都必须有该属性的值（NOT NULL）

(7)属性特点——#IMPLIED

这就表示该属性的值是可以被忽略的。

（8)属性特点——#FIXED

元素中该属性的值必须为指定的固定值。

（9）属性特点——Default

为属性提供一个默认的值。

## 三、xml之Schema技术

**1、Schema概述：**xml Schema是用一套预先规定的xml元素和属性创建的，这些元素和属性定义了xml文档的结构和内容模式。Xml Shema规定xml文档实例的结构和每个元素/属性的数据类型。

理解：对于Schema来讲，我们可以将shema比喻成为是表结构。在表结构里，定义一些数据的限制要求。然后我们xml文档呢就相当于是数据表将来要存储的数据u，也就是数据库里面的数据表数据了，所以整体来讲的话schema它就是用于验证xml文档的。

**2、有了DTD，为什么要用Schema呢？**

（1）DTD的局限性

DTD不遵守xml语法（写xml文档实例时用一种语法，写DTD的时候用另外一种语法）；DTD数据类型有限（与数据库数据类型不一致）；DTD不可扩展；DTD不支持命名空间（命名冲突）。

（2）Schema的新特性

Shema基于xml语法的；Shema可以用能处理xml文档的工具处理；Schema大大扩充了数据类型，可以自定义数据类型；Schema支持元素的继承——Object-Oriented;Shema支持属性组。

**3、Shema（模式）：**其作用与DTD一样，也是用于验证xml文档的有效性，只不过它提供了比DTD更强大的功能和更细粒度的数据类型，另外Schema还可以自定义数据类型。此外，Schema也是一个xml文件，而DTD则不是。

**4、所有的Schema文档，其根元素必须为Schema。**

**5、Schema的数据类型概述**

**6、复合类型（通过complexType定义）**

**7、Schema的元素类型**

**7.1、Schema根元素**

作用：包含已经定义的Schema

用法：<xs:schema>

属性：xmlns/targetNamespace/elementFormDefault

**7.2、element元素**

作用：声明一个元素

属性：name/type/ref/minOccurs/maxOccurs

**7.3、group元素**

作用：把一组元素声明组合在一起

属性:name

**7.4、attribute元素**

作用：声明一个属性

属性：name/type/use/default/fixed

**7.5、attributeGroup元素**

作用：把一组属性声明组合在一起

属性：name/ref

**7.6、simpleType元素**

作用：定义一个简单类型，它决定了元素和属性值的约束和相关信息

属性：name

常用的两种方式：restriction——>一个约束；list——>从列表中选择。

（1）simpleType元素的子元素——>restriction

子元素为：<xs:restriction>——>定义一个约束条件

（2）simpleType元素的子元素——>list

子元素为：<xs:list>——>从一个特定数据类型的集合中选择定义一个简单类型元素。

**7.7、complexType元素**

作用：定义一个复合类型，它决定了一组元素和属性值的约束和相关信息。

属性：name

常用的两种方式：sequence——>一个序列；choice——>设置选择项。

（1）complexType元素——sequence

作用：给一组元素一个特定的序列

（2）complexType元素——choice

作用：把一组属性声明组合在一起，以便可以被复合类型应用

属性：name/ref

**7.8、complexType与simpleType区别**

simpleType类型的元素中不能包含元素或者属性；

当需要声明一个元素的子元素和或属性时，用complexType；

当需要基于内置的基本数据类型定义一个新的数据类型时，用simpleType。

## 四、标签含义详解

**1 DTD比Schema**

可以使用DTD（文档类型定义）来规定xml文档构建模块，可用起对xml文件进行验证。同样也可以用Schema来实现DTD的功能，并且Schema因为其本身可以规定数据类型和其扩展性要优于DTD，并且将逐步取代DTD。

**2 xmlns 的作用**

一个 xml 文档中如果包含如下两种定义不同， 但是名称相同的元素， xml 解析器是无法解析的， 因为它不能确定当你调用 document.getElementsByTagName("book") 时应该返回哪个元素。

<table>

<tr>

<td>Apples</td>

<td>Bananas</td>

</tr>

</table>

<table>

<name>African Coffee Table</name>

<width>80</width>

<length>120</length>

</table>

显然， 如果给他们的名字添加一个前缀， 则命名冲突的问题就可以解决。

<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

<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>

命名空间就诞生了， 我们可以为元素定义一个命名空间， 将一个很长的， 可以保证全局唯一性的字符串与该元素关联起来。这样就可以避免命名冲突了。但是如何保证那个较长的字符串全局唯一呢， 最好的方式莫过于使用 统一资源标识符（Uniform Resource Identifier，URI) 了， 而我们最常见的 URI 就是平时经常访问的网址 URL 了。

xmlns:namespace-prefix="namespaceURI"

**3 project 元素前面并没有前缀**

<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://maven.apache.org/POM/4.0.0"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

这里实际上使用的是默认命名空间（default naming space）。使用默认命名空间的作用是， 该元素内部的子元素都会默认属于该命名空间下， 我们不需要为他们一一添加命名空间前缀。 它的语法如下：

<elementName xmlns="namespaceURI">

**4 xmlns:xsi**

xmlns:xsi 定义了一个命名空间前缀 xsi 对应的唯一字符串 http://www.w3.org/2001/XMLSchema-instance。 但是读者会发现， 这个 xmlns:xsi 在不同的 xml 文档中似乎都会出现。 这是因为， xsi 已经成为了一个业界默认的用于 XSD(（XML Schema Definition) 文件的命名空间。 而 XSD 文件（也常常称为 Schema 文件）是用来定义 xml 文档结构的。

**5 xsi:schemaLocation**

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"

上面这行的语法其实是， xsi:schemaLocation = "键" "值"。即 xsi 命名空间下 schemaLocation 元素的值为一个由空格分开的键值对。前一个“键” http://maven.apache.org/POM/4.0.0 指代 【命名空间】， 只是一个全局唯一字符串而已。后一个值指代 【XSD location URI】 , 这个值指示了前一个命名空间所对应的 XSD 文件的位置， xml parser 可以利用这个信息获取到 XSD 文件， 从而通过 XSD 文件对所有属于 命名空间 http://maven.apache.org/POM/4.0.0 的元素结构进行校验， 因此这个值必然是可以访问的， 且访问到的内容是一个 XSD 文件的内容

**6 实例解析**

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd"

default-lazy-init="true">

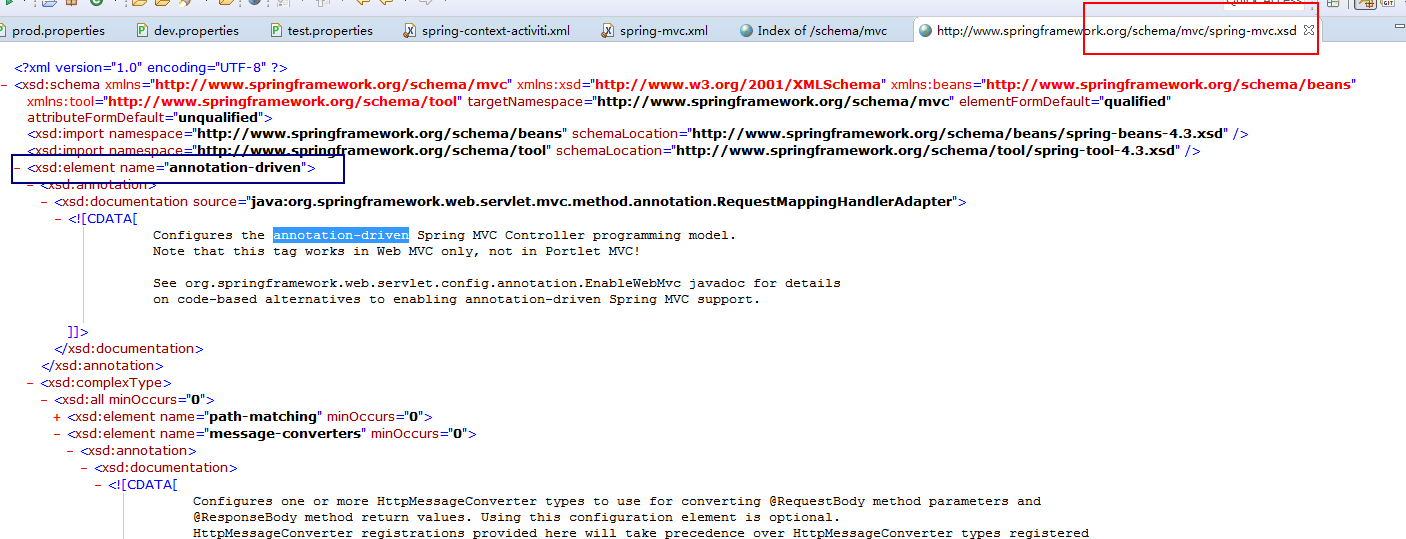
（1）beans使用默认命名空间，xmlns="http://www.springframework.org/schema/beans"

（2）xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"标识业界公认的W3C标准，里面定义了schemaLocation前缀

（3）xmlns:context表示context前缀的命名空间 ，比如<context:component-scan ../>

（4）xmlns:mvc=表示mvc前缀的命名空间 ，比如<mvc:interceptors>,<mvc:resources>,<mvc:annotation-driven>

（5）xsi:schemaLocation表示键值对，命名空间和校验文件xsd之间的映射。



**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。**XML的解析方式分为四种：1、DOM解析；2、SAX解析；3、JDOM解析；4、DOM4J解析。**其中前两种属于基础方法，是官方提供的平台无关的解析方式；后两种属于扩展方法，它们是在基础的方法上扩展出来的，只适用于java平台。

针对以下XML文件，会对四种方式进行详细描述：

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

<bookstore>

<book id="1">

<name>冰与火之歌</name>

<author>乔治马丁</author>

<year>2014</year>

<price>89</price>

</book>

<book id="2">

<name>安徒生童话</name>

<year>2004</year>

<price>77</price>

<language>English</language>

</book>

</bookstore>

## 一、DOM解析

**1 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。

**2 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。

**3 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。

**4 解析代码**

public class DOMTest {

public static void main(String[] args) {

//创建一个DocumentBuilderFactory的对象

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

//创建一个DocumentBuilder的对象

try {

//创建DocumentBuilder对象

DocumentBuilder db = dbf.newDocumentBuilder();

//通过DocumentBuilder对象的parser方法加载books.xml文件到当前项目下

Document document = db.parse("books.xml");

//获取所有book节点的集合

NodeList bookList = document.getElementsByTagName("book");

//通过nodelist的getLength()方法可以获取bookList的长度

System.out.println("一共有" + bookList.getLength() + "本书");

//遍历每一个book节点

for (int i = 0; i < bookList.getLength(); i++) {

System.out.println("=================下面开始遍历第" + (i + 1) + "本书的内容=================");

//通过 item(i)方法 获取一个book节点，nodelist的索引值从0开始

Node book = bookList.item(i);

//获取book节点的所有属性集合

NamedNodeMap attrs = book.getAttributes();

System.out.println("第 " + (i + 1) + "本书共有" + attrs.getLength() + "个属性");

//遍历book的属性

for (int j = 0; j < attrs.getLength(); j++) {

//通过item(index)方法获取book节点的某一个属性

Node attr = attrs.item(j);

//获取属性名

System.out.print("属性名：" + attr.getNodeName());

//获取属性值

System.out.println("--属性值" + attr.getNodeValue());

}

//解析book节点的子节点

NodeList childNodes = book.getChildNodes();

//遍历childNodes获取每个节点的节点名和节点值

System.out.println("第" + (i+1) + "本书共有" +

childNodes.getLength() + "个子节点");

for (int k = 0; k < childNodes.getLength(); k++) {

//区分出text类型的node以及element类型的node

if (childNodes.item(k).getNodeType() == Node.ELEMENT\_NODE) {

//获取了element类型节点的节点名

System.out.print("第" + (k + 1) + "个节点的节点名："

+ childNodes.item(k).getNodeName());

//获取了element类型节点的节点值

System.out.println("--节点值是：" + childNodes.item(k).getFirstChild().getNodeValue());

//System.out.println("--节点值是：" + childNodes.item(k).getTextContent());

}

}

System.out.println("======================结束遍历第" + (i + 1) + "本书的内容=================");

}

} catch (ParserConfigurationException e) {

e.printStackTrace();

} catch (SAXException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

## 二、SAX解析

**1 SAX的全称是Simple APIs for XML，也即XML简单应用程序接口。**与DOM不同，SAX提供的访问模式是一种顺序模式，这是一种快速读写XML数据的方式。

当使用SAX分析器对XML文档进行分析时，会触发一系列事件，并激活相应的事件处理函数，应用程序通过这些事件处理函数实现对XML文档的访问，因而

SAX接口也被称作事件驱动接口。

**2 优点：**

（1）采用事件驱动模式，对内存耗费比较小。

（2）适用于只处理XML文件中的数据时。

**3 缺点：**

（1）编码比较麻烦。

（2）很难同时访问XML文件中的多处不同数据。

**4 以下是解析代码**

public class SAXTest {

public static void main(String[] args) {

//锟斤拷取一锟斤拷SAXParserFactory锟斤拷实锟斤拷

SAXParserFactory factory = SAXParserFactory.newInstance();

//通锟斤拷factory锟斤拷取SAXParser实锟斤拷

try {

SAXParser parser = factory.newSAXParser();

//锟斤拷锟斤拷SAXParserHandler锟斤拷锟斤拷

SAXParserHandler handler = new SAXParserHandler();

parser.parse("books.xml", handler);

System.out.println("~！~！~！共有" + handler.getBookList().size()

+ "本书");

for (Book book : handler.getBookList()) {

System.out.println(book.getId());

System.out.println(book.getName());

System.out.println(book.getAuthor());

System.out.println(book.getYear());

System.out.println(book.getPrice());

System.out.println(book.getLanguage());

System.out.println("----finish----");

}

} catch (ParserConfigurationException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (SAXException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

public class SAXParserHandler extends DefaultHandler {

String value = null;

Book book = null;

private ArrayList<Book> bookList = new ArrayList<Book>();

public ArrayList<Book> getBookList() {

return bookList;

}

int bookIndex = 0;

/\*\*

\* 用来标识解析开始

\*/

@Override

public void startDocument() throws SAXException {

// TODO Auto-generated method stub

super.startDocument();

System.out.println("SAX解析开始");

}

/\*\*

\* 用来标识解析结束

\*/

@Override

public void endDocument() throws SAXException {

// TODO Auto-generated method stub

super.endDocument();

System.out.println("SAX解析结束");

}

/\*\*

\* 解析xml元素

\*/

@Override

public void startElement(String uri, String localName, String qName,

Attributes attributes) throws SAXException {

//调用DefaultHandler类的startElement方法

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

if (qName.equals("book")) {

bookIndex++;

//创建一个book对象

book = new Book();

//开始解析book元素的属性

System.out.println("======================开始遍历某一本书的内容=================");

//不知道book元素下属性的名称以及个数，如何获取属性名以及属性值

int num = attributes.getLength();

for(int i = 0; i < num; i++){

System.out.print("book元素的第" + (i + 1) + "个属性名是："

+ attributes.getQName(i));

System.out.println("---属性值是：" + attributes.getValue(i));

if (attributes.getQName(i).equals("id")) {

book.setId(attributes.getValue(i));

}

}

}

else if (!qName.equals("name") && !qName.equals("bookstore")) {

System.out.print("节点名是：" + qName + "---");

}

}

@Override

public void endElement(String uri, String localName, String qName)

throws SAXException {

//调用DefaultHandler类的endElement方法

super.endElement(uri, localName, qName);

//判断是否针对一本书已经遍历结束

if (qName.equals("book")) {

bookList.add(book);

book = null;

System.out.println("======================结束遍历某一本书的内容=================");

}

else if (qName.equals("name")) {

book.setName(value);

}

else if (qName.equals("author")) {

book.setAuthor(value);

}

else if (qName.equals("year")) {

book.setYear(value);

}

else if (qName.equals("price")) {

book.setPrice(value);

}

else if (qName.equals("language")) {

book.setLanguage(value);

}

}

@Override

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

throws SAXException {

// TODO Auto-generated method stub

super.characters(ch, start, length);

value = new String(ch, start, length);

if (!value.trim().equals("")) {

System.out.println("节点值是：" + value);

}

}

}

## 三、JDOM解析

**1特征：**

（1）仅使用具体类，而不使用接口。

（2）API大量使用了Collections类。

**2 以下是解析代码：**

public class JDOMTest {

private static ArrayList<Book> booksList = new ArrayList<Book>();

public static void main(String[] args) {

// 进行对books.xml文件的JDOM解析

// 准备工作

// 1.创建一个SAXBuilder的对象

SAXBuilder saxBuilder = new SAXBuilder();

InputStream in;

try {

// 2.创建一个输入流，将xml文件加载到输入流中

in = new FileInputStream("src/res/books.xml");

InputStreamReader isr = new InputStreamReader(in, "UTF-8");

// 3.通过saxBuilder的build方法，将输入流加载到saxBuilder中

Document document = saxBuilder.build(isr);

// 4.通过document对象获取xml文件的根节点

Element rootElement = document.getRootElement();

// 5.获取根节点下的子节点的List集合

List<Element> bookList = rootElement.getChildren();

// 继续进行解析

for (Element book : bookList) {

Book bookEntity = new Book();

System.out.println("======开始解析第" + (bookList.indexOf(book) + 1)

+ "书======");

// 解析book的属性集合

List<Attribute> attrList = book.getAttributes();

// //知道节点下属性名称时，获取节点值

// book.getAttributeValue("id");

// 遍历attrList(针对不清楚book节点下属性的名字及数量)

for (Attribute attr : attrList) {

// 获取属性名

String attrName = attr.getName();

// 获取属性值

String attrValue = attr.getValue();

System.out.println("属性名：" + attrName + "----属性值："

+ attrValue);

if (attrName.equals("id")) {

bookEntity.setId(attrValue);

}

}

// 对book节点的子节点的节点名以及节点值的遍历

List<Element> bookChilds = book.getChildren();

for (Element child : bookChilds) {

System.out.println("节点名：" + child.getName() + "----节点值："

+ child.getValue());

if (child.getName().equals("name")) {

bookEntity.setName(child.getValue());

}

else if (child.getName().equals("author")) {

bookEntity.setAuthor(child.getValue());

}

else if (child.getName().equals("year")) {

bookEntity.setYear(child.getValue());

}

else if (child.getName().equals("price")) {

bookEntity.setPrice(child.getValue());

}

else if (child.getName().equals("language")) {

bookEntity.setLanguage(child.getValue());

}

}

System.out.println("======结束解析第" + (bookList.indexOf(book) + 1)

+ "书======");

booksList.add(bookEntity);

bookEntity = null;

System.out.println(booksList.size());

System.out.println(booksList.get(0).getId());

System.out.println(booksList.get(0).getName());

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (JDOMException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

## 四、DOM4J解析

**1 特征：**

（1）JDOM的一种智能分支，它合并了许多超出基本XML文档表示的功能。

（2）它使用接口和抽象基本类方法。

（3）具有性能优异、灵活性好、功能强大和极端易用的特点。

（4）是一个开放源码的文件

**2 以下是解析代码：**

public class DOM4JTest {

private static ArrayList<Book> bookList = new ArrayList<Book>();

public static void main(String[] args) {

// 解析books.xml文件

// 创建SAXReader的对象reader

SAXReader reader = new SAXReader();

try {

// 通过reader对象的read方法加载books.xml文件,获取docuemnt对象。

Document document = reader.read(new File("src/res/books.xml"));

// 通过document对象获取根节点bookstore

Element bookStore = document.getRootElement();

// 通过element对象的elementIterator方法获取迭代器

Iterator it = bookStore.elementIterator();

// 遍历迭代器，获取根节点中的信息（书籍）

while (it.hasNext()) {

System.out.println("=====开始遍历某一本书=====");

Element book = (Element) it.next();

// 获取book的属性名以及 属性值

List<Attribute> bookAttrs = book.attributes();

for (Attribute attr : bookAttrs) {

System.out.println("属性名：" + attr.getName() + "--属性值："

+ attr.getValue());

}

Iterator itt = book.elementIterator();

while (itt.hasNext()) {

Element bookChild = (Element) itt.next();

System.out.println("节点名：" + bookChild.getName() + "--节点值：" + bookChild.getStringValue());

}

System.out.println("=====结束遍历某一本书=====");

}

} catch (DocumentException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

## 五、比较总结

**1 DOM4J性能最好，连Sun的JAXM也在用DOM4J。**目前许多开源项目中大量采用DOM4J，例如大名鼎鼎的Hibernate也用DOM4J来读取XML配置文件。如果不考虑可移植性，那就采用DOM4J。

2 **JDOM和DOM在性能测试时表现不佳，在测试10M文档时内存溢出。**在小文档情况下还值得考虑使用DOM和JDOM。虽然JDOM的开发者已经说明他们期望在正式发行版前

专注性能问题，但是从性能观点来看，它确实没有值得推荐之处。另外，DOM仍是一个非常好的选择。DOM实现广泛应用于多种编程语言。它还是许多其它与XML相关的

标准的基础，因为它正式获得W3C推荐（与基于非标准的Java模型相对），所以在某些类型的项目中可能也需要它（如在JavaScript中使用DOM）。

3 **SAX表现较好，这要依赖于它特定的解析方式－事件驱动。**一个SAX检测即将到来的XML流，但并没有载入到内存（当然当XML流被读入时，会有部分文档暂时隐藏在内存中）。