**XML的作用**

1. 一种文档格式.只是内容的载体.
2. 常用来做数据存储,数据传输或者配置描述.
3. 它不负责展示.至于里面的内容如何使用,由XML程序来控制.

**XML的格式**

1. 首先第一行为XML的声明:

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

|  |  |
| --- | --- |
|  |  |

1. 紧跟着可能会有DTD校验方法.

<!DOCTYPE root-element SYSTEM "filename">

1. 如果XML想依托工具自动展现,需要XML展现方法. CSS或者XSLT.

<?xml-stylesheet type="text/css" href="cd\_catalog.css"?>

或者

<?xml-stylesheet type="text/xsl" href="simple.xsl"?>

1. Element所构成的树形结构.
2. Element上的namespace.
3. 除了用DTD验证方法,也可以Element上使用XSD来校验XML的合法性.

<note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.w3schools.com note.xsd">

 ...

</note>

**XML字符编码**

XML存储时所使用的字符编码. 这个编码告诉解析程序应该使用什么编码格式来对XML解码. 为了国际通用,使用UTF-8吧. 对于纯英文,UTF8只需要一个字节来表示一个英文字符. XML的size也不会太大.

**XML命名空间**

命名空间语法包括声明部分 默认命名xmlns="<URL>"或者指定命名xmlns:prefix="http://<namespace specification URL>" 和 使用部分<prefix:tag>或者<tag prefix:attr="">.

命名空间解决了两个问题.

1. 相同名称的标签表示不同的意义,它们各自存在与自己的命名空间中.比如<table>即可以表示表格,也可以表示桌子. 给他们一个命名空间. <n1:table>为表单,<n2:table>为桌子.
2. 对既有的元素进行属性扩展或者元素扩展. 比如本文例子中的<book>多了audlt的属性和子元素.它是对原来元素的扩展.

**在Java或者JavaScript中是使用namespace的, 注意以下几点:**

1. DOM中存在两个方法getElementsByTagName()和getElementsByTagNameNS(). 第一个方法需要使用qualified name作为参数,而第二个方法需要使用namespace和localname作为参数. 如下

document.getElementsByTagNameNS("http://japan.org/book/audlt", "age");

document.getElementsByTagName("audlt:age");

1. 如果XML里面使用了namespace, 那么XSLT和XPATH也必须使用同等的namespace,否则xpath将搜索不到你想查找的元素,在java的Xpath中,需要设置NamespaceContext. 请看DOM实例和我写的XSL文件.

## XML语法验证

验证XML合法性靠的是DTD或者XSD.这是XML的两个规范. XSD比DTD要新,所以也先进.

### DTD

本文中的XML里面声明了DTD的引用,XML parser就会自动加载DTD来验证XML. 这需要给parser设定两个前提.一是开启了验证模式,而是明白DTD的加载位置. XML parser可以是JS,java或者browser. 加载位置可以使用PUBLIC ID或者SYSTEM ID来判断.请看下面的声明:

<!DOCTYPE bookStore SYSTEM "bookStore.dtd">

上面的声明没有PUBLIC ID, 只有SYSTEM ID, SYSTEM ID=XML当前路径+"/bookStore.dtd". 可见system id是一个相对与XML的路径.

声明PUBLIC ID:

<!DOCTYPE bookStore PUBLIC "bookStore.dtd" "bookStore.dtd">

|  |  |
| --- | --- |
|  |  |

PUBLIC ID也为"bookStore.dtd". 这时候,Parser会自动根据这两个ID去尝试加载DTD文件,如果加载不到,则抛出exception. JAVA中,我们可以通过实现EntityResolver接口的方法来自定义DTD的所在位置. 详情请看JAVA部分.

本文用的DTD是:

<!ELEMENT bookStore (keeper, books)>

<!ATTLIST bookStore name CDATA #REQUIRED>

<!ELEMENT keeper (name)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT books (book)>

<!ELEMENT book (title, author)>

<!ATTLIST book id ID #REQUIRED>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (#PCDATA)>

### XSD

使用XSD来验证XML只需要一个XSD的定义文件,开启Parser的XSD验证功能. XSD的验证方法在后面的JAVA代码中可以看到. 本文使用的XSD如下:

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

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

    <xsd:element name="bookStore" type="bookStoreType" />

    <xsd:complexType name="bookStoreType">

        <xsd:sequence>

            <xsd:element name="keeper" type="keeperType"></xsd:element>

            <xsd:element name="books" type="booksType"></xsd:element>

        </xsd:sequence>

        <xsd:attribute name="name" type="xsd:string"></xsd:attribute>

    </xsd:complexType>

    <xsd:complexType name="keeperType">

        <xsd:sequence>

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

        </xsd:sequence>

    </xsd:complexType>

    <xsd:complexType name="booksType">

        <xsd:sequence>

            <xsd:element name="book" type="bookType"></xsd:element>

        </xsd:sequence>

    </xsd:complexType>

    <xsd:complexType name="bookType">

        <xsd:sequence>

            <xsd:element name="title" type="xsd:string"></xsd:element>

            <xsd:element name="author" type="xsd:string"></xsd:element>

        </xsd:sequence>

        <xsd:attribute name="id" type="xsd:int"></xsd:attribute>

    </xsd:complexType>

</xsd:schema>

## XML展示方法(CSS, XSL)

如下面的代码片段所示,XML可以有stylesheet转换成其他格式, 如HTML, TXT等. stylesheet可以是css,也可以是xsl.

<?xml-stylesheet type="test/xsl" href="bookStore.xsl"?>

主流browser都已经支持这种转换格式. 除了自动转换,我们也可以使用代码对转换进行控制.我们可以用java在服务器端进行xslt的转换,也可以使用javascript在前端对xml进行xslt转换. 代码在后面均可找到.  书写xsl的时候,namespace一定要注意. xpath一定要和namespace所对应. 我所使用的**XSL**为:

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

<xsl:stylesheet version="1.0"

    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:b="http://joey.org/bookStore"

    xmlns:a="http://japan.org/book/audlt">

    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"></xsl:output>

    <xsl:template match="/">

        <html>

            <body>

                <h2>Book Store&lt;&lt;<xsl:value-of select="/b:bookStore/@name"></xsl:value-of>&gt;&gt;</h2>

                <div>

                    There are <xsl:value-of select="count(/b:bookStore/b:books/b:book)"></xsl:value-of> books.

                </div>

                <div>

                    Keeper of this store is <xsl:value-of select="/b:bookStore/b:keeper/b:name"></xsl:value-of>

                </div>

                <xsl:for-each select="/b:bookStore/b:books/b:book">

                    <div> Book:

                        <span>title=<xsl:value-of select="b:title"></xsl:value-of></span>;

                        <span>author=<xsl:value-of select="b:author"></xsl:value-of></span>

                        <xsl:if test="@a:color">

                            <span style="color:yellow">H Book, require age<xsl:value-of select="a:age"></xsl:value-of></span>

                        </xsl:if>

                    </div>

                </xsl:for-each>

            </body>

        </html>

    </xsl:template>

</xsl:stylesheet>

## XML与javascript

Javascript对XML的支持在IE和FF+Chrome上是不同的. IE使用的ActiveXObject来生成一个XML的实例.FF与Chrome等其它主流浏览器均遵循w3c规范. 生成的XML document可以使用其DOM方法对dom tree进行操作. 也可以借助框架dojo,jquery等简化操作.

下面这个例子是使用JS对XML进行XSLT转化,从而生成HTML.

function createXMLDoc(xmlStr) {

    var xmlDoc;

    if (window.DOMParser) {

        // FF Chrome

        var parser=new DOMParser();

        xmlDoc=parser.parseFromString(xmlStr,"text/xml");

    } else if (window.ActiveXObject){

        // Internet Explorer

        xmlDoc=new ActiveXObject("Microsoft.XMLDOM");

        xmlDoc.async="false";

        xmlDoc.loadXML(xmlStr);

    }

    return xmlDoc;

}

function transform(xmlDoc, xslDoc) {

    if (window.XSLTProcessor) {

        // chrome FF

        var xslp = new XSLTProcessor();

        xslp.importStylesheet(xslDoc);

        return xslp.transformToFragment(xmlDoc,document);

    } else if (window.ActiveXObject){

        // IE

        return xmlDoc.transformNode(xslDoc);

    }

}

var xmlStr =

    ['<bookStore name="java" xmlns="http://joey.org/bookStore" xmlns:audlt="http://japan.org/book/audlt" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="bookStore.xsd">',

       '<keeper><name>Joey</name></keeper>',

       '<books>',

         '<book id="1"> <title>XML</title><author>Steve</author></book>',

         '<book id="2"><title>JAXP</title> <author>Bill</author></book>',

         '<book id="3" audlt:color="yellow"><audlt:age> &gt;18 </audlt:age> <title>Love</title><author>teacher</author></book>',

       '</books></bookStore>'].join('');

var xslStr =

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

       '<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:b="http://joey.org/bookStore" xmlns:a="http://japan.org/book/audlt">',

       '<xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes" />',

       '<xsl:template match="/">',

         '<html>',

          '<body>',

             '<h2>Book Store&lt;&lt;<xsl:value-of select="/b:bookStore/@name"/>&gt;&gt;</h2>',

             '<div>There are <xsl:value-of select="count(/b:bookStore/b:books/b:book)"/> books.</div>',

             '<div>Keeper of this store is <xsl:value-of select="/b:bookStore/b:keeper/b:name"/></div>',

             '<xsl:for-each select="/b:bookStore/b:books/b:book">',

             '<div>Book: ',

             '<span>title=<xsl:value-of select="b:title"/></span>;<span>author=<xsl:value-of select="b:author"/></span>',

             '<xsl:if test="@a:color">',

             '<span color="yellow">H Book, require age<xsl:value-of select="a:age"/></span>',

             '</xsl:if>',

             '</div>',

             '</xsl:for-each>',

          '</body>',

         '</html>',

       '</xsl:template>',

      '</xsl:stylesheet>'].join('');

var xmlDoc = createXMLDoc(xmlStr);

var xslDoc = createXMLDoc(xslStr);

var dom = transform(xmlDoc, xslDoc);

console.log(dom.childNodes[0].outerHTML);

## XML与java

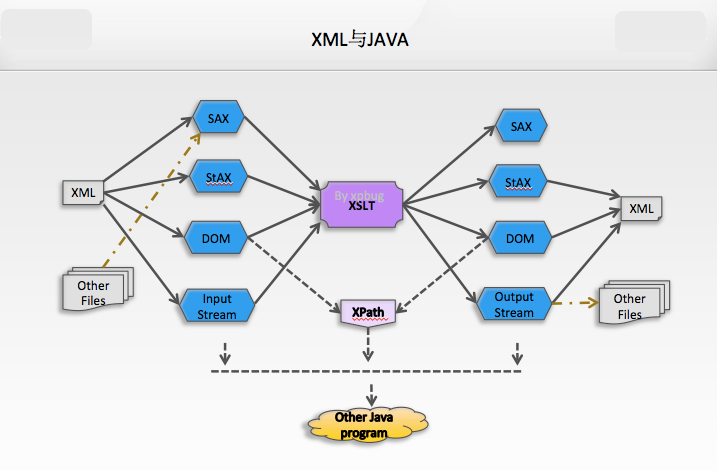
Java对XML的支持被称为JAXP(Java API for XML Processing). JAXP被当做标准,放入了J2SE1.4.从此以后,JRE自带XML的处理类库. 当然,JAXP允许使用第三方的XML Parser,不同的parser有着不同的优缺点,用户可以自己选择. 但所有的Parser均必须实现JAXP所约定的Interface. **掌握JAXP,需要知道以下内容. 这些都会在后面进行描述.**

1. **JAXP的parser以及如何使用第三方parser.**
2. **XML的解析方法SAX,DOM以及STAX.**
3. **XML的写出方法STAX和XSLT.**
4. **使用XPath搜索DOM.**
5. **JAXP使用XSLT转换XML.**
6. **DOM与JDOM,DOM4J的区别.**
7. **JAXP验证XML.**
8. **JAXP支持namespace**

J2SE的JAXP提供了5个包,用于支持XML.

1. javax.xml.parsers - 为各种第三方parser提供了接口.
2. org.w3c.dom - 提供了DOM类
3. org.xml.sax - 提供了SAX类
4. javax.xml.transform - 提供了XSLT的API.
5. javax.xml.stream - 提供了STAX的API. STAX比SAX简单,比DOM快.
6. javax.xml.xpath - 使用xpath对DOM进行字段查询.

每个接口与类的使用方法就不使用文字描述了,后面会用代码和注释的方式一一介绍JAXP的类库. 在描述SAX,StAX,DOM等方法之前,有必要做一个highlevel的比较. 每一个解析方法的优缺点是什么?改如何选择它们.

首先,XML解析器存在SAX, StAX和DOM, 而XML文件生成方法又有StAX和DOM. XPath是一个查询DOM的工具. XSLT是转换XML格式的工具. 如下图所示:[](http://static.oschina.net/uploads/space/2013/0124/163110_B3K1_254689.png)

XML的解析从数据结构上来讲,分两大类: Streaming和Tree. Streaming又分为SAX和StAX. Tree就是DOM. SAX和StAX均是顺序解析XML,并生成读取事件.我们可以通过监听事件来得到我们想要的内容. DOM是一次性的以tree结构形式载入内存.

Streaming VS DOM

1. DOM需要内存.对于大文档或者多文档,DOM性能差.还有,在android手机上就少用DOM这种占内存的东东吧.
2. Streaming是实时性的,它没有上下文. 如果一个XML的element需要上下文才能理解,使用DOM会方便.
3. 如果XML来自网络,我们对其结构并不明朗,使用Streaming比较好. DOM适合对XML的结构非常清楚.比如web.xml的结构就是一个人人皆知的结构.
4. 需要对XML进行增删改查.则使用DOM.

Streaming又包含SAX和StAX, SAX是推(push)解析方法,而StAX是拉(pull)解析方法. 后面有SAX和StAX的实例.

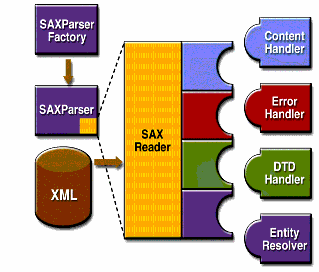
Pull VS Push

1. Pull可以让我们的代码掌握主动权,在合适的时候去调用解析器继续工作. Push是被动的听从解析器只会.解析器会不停的读,并把事件push到handler中.
2. Pull的代码简单,小.Lib也小.
3. Pull可以一个线程同时解析多个文档. 因为主动权在我们.
4. StAX可以将一个普通的数据流伪造成一个个XML的读取事件,从而在构造成一个XML.好似DB中的View.

|  |  |  |  |
| --- | --- | --- | --- |
|  | SAX | StAX | DOM |
| API Type | Push, Streaming | Pull, Streaming | Tree, In momery |
| Support XPath? | No | No | Yes |
| Read XML | Yes | Yes | Yes |
| Write XML | No | Yes | Yes |
| CRUD | No | No | Yes |
| Parsing Validation  (DTD, XSD) | Yes | Optional (JDK embedded  Parser does not support it). | Yes |

javax.xml.validation包提供了跟XML解析独立与解析过程的验证方法. 性能比不过Parsing Validation. Parsing validation指的是在解析过程中进行验证.

### SAX实例

借用oracle网上的一张图来说明SAX的架构.  
[](http://static.oschina.net/uploads/space/2013/0119/140014_aGIb_254689.gif)

SAXParser是调用XMLReader的, 如果使用SAXParser,则需要传参DefaultHandler. DefaultHandler实现了上图的4个Handler接口. 你也可以直接使用XMLReader,然后调用它的parser方法.只是在parser前,需set每个Handler. SAXParser是Event-Driven设计模式, 随着读取XML的字节,随着传递event给handler来处理.

读的工作其实是有XMLReader来做的,所有的events也是XMLReader产生的.所以,将一个非XML格式的文件模拟成一个XML,只需要 复写XMLReader,读取非XML文件时,发出假的Event,这样handler将会把这个文件当做一个XML来处理. 这种机制会在XSLT中用到.

**关于模拟XML**

SAX可以将一个非XML格式文件的读取模拟成一个XML的文件的读取.通过构造XML的读取Event. 只是SAX需要复写XMLReader.

#### ContentHandler

用于处理XML的各种数据类型的读取事件.这里面的事件有

1. setDocumentLocator. 读取<?xml ...?>
2. startDocument and endDocument. XML的最外层tag的开始与结束.
3. startPrefixMapping and endPrefixMapping. 命名空间影响范围的进入与退出.
4. startElement and endElement. 每个Element的开始与结束.
5. characters. 读取Element的text node value.

实现方式可以参考org.xml.sax.helpers.DefaultHandler.

#### ErrorHandler

用于处理XML解析阶段所发生的警告和错误.里面有三个方法,warning(), error()和fatalError(). waring和error用于处理XML的validation(DTD或XSD)错误.这种错误并不影响XML的解析,你可以把这种错误产生的 exception压下来,而不向上抛.这样XML的解析不会被终断. fatalError是XML结构错误,这种错误无法被压制,即使我的handler不抛,Parser会向外抛exception.

#### DTDHandler

DTD定义中存在ENTITY和NOTATION.这都属于用户自定义属性. XML Parser无法理解用户自定义的ENTITY或者NOTATION, 于是它把这方面的验证工作交给了DTDHandler. DTDHandler里面只有2个方法:notationDecl和unparsedEntityDecl. 我们实现这两个方法来验证我们的NOTATION部分是否正确.

#### EntityResolver

在XML的验证段落里面提到过DTD的定位. EntityResolver可以帮助我们做这件事情. EntityResolver里面只有一个方法,叫做ResolveEntity(publicId, systemId). 每当Parser需要使用external文件的时候,就会调用这个方法. 我们可以在这个方法里面做一些预处理. 代码如下:

public class MyEntityResolver implements EntityResolver {

    @Override

    public InputSource resolveEntity(String publicId, String systemId)

            throws SAXException, IOException {

        if ("bookStore.dtd".equals(publicId)) {

            InputStream in = this.getClass().getResourceAsStream("/jaxp/resources/bookStore.dtd");

            InputSource is = new InputSource(in);

            return is;

        }

        return null;

    }

}

#### SAX Parser的使用

请注意里面是如何开启validation模式的. XSD有两种开启方法.

public class MySAX {

    private SAXParser parser;

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

        new MySAX();

    }

    public MySAX() throws ParserConfigurationException, SAXException, IOException {

        // Use "javax.xml.parsers.SAXParserFactory" system property to specify a Parser.

        // java -Djavax.xml.parsers.SAXParserFactory=yourFactoryHere [...]

        // If property is not specified, use J2SE default Parser.

        // The default Parser is "com.sun.org.apache.xerces.internal.jaxp.SAXParserFactoryImpl".

        SAXParserFactory spf = SAXParserFactory.newInstance();

        spf.setNamespaceAware(true);

        // Use XSD defined by JAXP 1.3, JAVA1.5

        //SchemaFactory sf = SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");

        //spf.setSchema(sf.newSchema(this.getClass().getResource("/jaxp/resources/bookStore.xsd")));

        // or Use old way defined by JAXP 1.2

        // parser.setProperty("http://java.sun.com/xml/jaxp/properties/schemaLanguage","http://www.w3.org/2001/XMLSchema");

        // parser.setProperty("http://java.sun.com/xml/jaxp/properties/schemaSource", new File("schema.xsd"));

<p>

// XSD disabled, use DTD. <span style="font-size:9pt;line-height:1.5;"> spf.setValidating(true);</span><span style="font-size:9pt;line-height:1.5;"> </span><span style="font-size:9pt;line-height:1.5;"> this.parser = spf.newSAXParser();</span>

</p>

// You can directly use SAXParser to parse XML. Or use XMLReader.

        // SAXParser warps and use XMLReader internally.

        // I will use XMLReader here.

        //this.parser.parse(InputStrean, DefaultHandler);

        XMLReader reader = this.parser.getXMLReader();

        reader.setContentHandler(new MyContentHandler());

        reader.setDTDHandler(new MyDTDHandler());

        reader.setErrorHandler(new MyErrorHandler());

        reader.setEntityResolver(new MyEntityResolver());

        InputStream in = this.getClass().getResourceAsStream("/jaxp/resources/bookStore.xml");

        InputSource is = new InputSource(in);

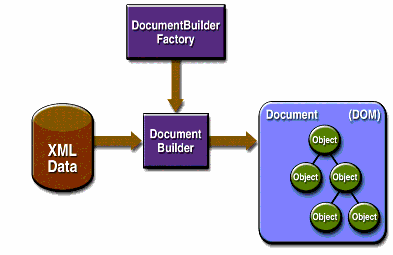
        is.setEncoding("UTF-8");

        reader.parse(is);

    }

}

### DOM实例 + XPath

借用oracle的图片来说明DOM解析的架构.  
[](http://static.oschina.net/uploads/space/2013/0119/171638_Z94Q_254689.gif)

JAVA对XML的解析标准存在DOM, JDOM, DOM4J. 有人认为JDOM和DOM4J都是DOM的另一种实现方法,这是错误的.

1. DOM是XML的数据模型标准,它跨越java,javascript等一切语言和平台.
2. JDOM和DOM4J是专门针对java的 模型.它简化了DOM,更加容易使用. 比如DOM中可以包含混合元素,即<a>text<b>text</b>test</a>. JDOM和DOM4J只允许<a>text</a>. 此外,DOM的数据访问模型也非常的复杂. 如果你的XML结构简单,可以使用JDOM和DOM4J. DOM4J的性能最好.

这篇文章只讲一下DOM. DOM的code和SAX的code相似的地方有:

1. 开启DTD或者XSD validation的方法.
2. 都用到ErrorHandler处理parser error和EntityResolver处理external引用.
3. 使用SAXException.但这都不意味着DomBuilder内部使用了SAXParser.

得到DOM数据模型以后，可以使用DOM的遍历方法来寻找元素，也可以使用**XPATH**来查找指定元素,XPath的重点注意事项是NamespaceContext.  接下来是DOM的code实例.

public class MyDOM {

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

        new MyDOM();

    }

    public MyDOM() throws Exception {

        // Use "javax.xml.parsers.DocumentBuilderFactory" system property to specify a Parser.

        // java -Djavax.xml.parsers.DocumentBuilderFactory=yourFactoryHere [...]

        // If property is not specified, use J2SE default Parser.

        // The default Parser is "com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl".

        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

        dbf.setIgnoringComments(false);

        dbf.setNamespaceAware(true);

        dbf.setIgnoringElementContentWhitespace(true);

        // Use XSD defined by JAXP 1.3, JAVA1.5

        // SchemaFactory sf = SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");

        // dbf.setSchema(sf.newSchema(this.getClass().getResource("/jaxp/resources/bookStore.xsd")));

        // or Use old way defined by JAXP 1.2

        // dbf.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage","http://www.w3.org/2001/XMLSchema");

        // dbf.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource", new File("schema.xsd"));

        // dbf.setSchema(schema);

        // XSD disabled, use DTD.

        dbf.setValidating(true);

        DocumentBuilder db = dbf.newDocumentBuilder();

        db.setErrorHandler(new MyErrorHandler());

        db.setEntityResolver(new MyEntityResolver());

        Document document = db.parse(this.getClass().getResourceAsStream("/jaxp/resources/bookStore.xml"));

        // Operate on Document according to DOM module.

        NodeList list = document.getElementsByTagNameNS("http://joey.org/bookStore", "book");

        System.out.println(list.item(2).getAttributes().item(0).getLocalName());

        // Node that if you don't specify name space, you need to use Qualified Name.

        System.out.println(document.getElementsByTagName("audlt:age").item(0).getTextContent());

        // Use xpath to query xml

        XPathFactory xpf = XPathFactory.newInstance();

        XPath xp = xpf.newXPath();

        // Need to set a namespace context.

        NamespaceContext nc = new NamespaceContext() {

            @Override

            public String getNamespaceURI(String prefix) {

                if (prefix.equals("b")) return "http://joey.org/bookStore";

                if (prefix.equals("a")) return "http://japan.org/book/audlt";

                return null;

            }

            @Override

            public String getPrefix(String namespaceURI) {

                if (namespaceURI.equals("http://joey.org/bookStore")) return "b";

                if (namespaceURI.equals("http://japan.org/book/audlt")) return "a";

                return null;

            }

            @Override

            public Iterator getPrefixes(String namespaceURI) {

                return null;

            }

        };

        xp.setNamespaceContext(nc);

        System.out.println(xp.evaluate("/b:bookStore/@name", document));

        System.out.println(xp.evaluate("/b:bookStore/b:books/b:book[@id=3]/@a:color", document));

    }

}

**StAX实例**

StAX和SAX比较,代码简单,且可以写XML. 但StAX规范对于解析时的validation不是强制的.所以,JDK自带StAX解析器就不支持Parsing Validation.

StAX存在两种API, Cursor API(XMLStreamReader, XMLStreamWriter)和Iterator API(XMLEventReader, XMLEventWriter). Cursor API就是一个像游标一样的读或者写API. 我们得不停的调用XML writer和XML reader来读写XML每一个字段,这是的代码逻辑层和XML解析层交叉在一起,很混乱. Iterator API将逻辑层和XML解析层分离,对Event进行封装,所有的数据都封装在Event中,逻辑层和解析层靠Event实体来打交道,实现了松耦合. 这是我的理解:

1. Cursor API比Iterator API更底层.
2. Iterator API对Event封装的比较好,隔离了逻辑层和XML解析层.实现了松耦合.逻辑层只需要focus在event数据本身上.
3. Iterator API更简单.推荐使用.
4. 使用Iterator API很容易实现将普通文本格式的内容伪装转化成一个XML格式的文件.

下面代码分别用Cursor API和Iterator API对XML解析,然后再重新生成写到JAVA Console.

**package** com.train.jaxp;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** javax.xml.namespace.NamespaceContext;

**import** javax.xml.namespace.QName;

**import** javax.xml.stream.XMLEventReader;

**import** javax.xml.stream.XMLEventWriter;

**import** javax.xml.stream.XMLInputFactory;

**import** javax.xml.stream.XMLOutputFactory;

**import** javax.xml.stream.XMLResolver;

**import** javax.xml.stream.XMLStreamConstants;

**import** javax.xml.stream.XMLStreamException;

**import** javax.xml.stream.XMLStreamReader;

**import** javax.xml.stream.XMLStreamWriter;

**import** javax.xml.stream.events.XMLEvent;

**import** com.train.jaxp.handle.MyXMLResolver;

**public** **class** MyStAX {

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

// coursorAPIReadWrite();

*eventAPIReadWrite*();

}

// use cursor API to read and write XML.

**public** **static** **void** coursorAPIReadWrite() **throws** Exception {

XMLInputFactory xif = XMLInputFactory.*newInstance*();

// Set properties for validation, namespace...

// But, JDK embeded StAX parser does not support validation.

//xif.setProperty(XMLInputFactory.IS\_VALIDATING, true);

xif.setProperty(XMLInputFactory.***IS\_NAMESPACE\_AWARE***, **true**);

// Handle the external Entity.

xif.setXMLResolver(**new** XMLResolver() {

**public** Object resolveEntity(String publicID, String systemID,

String baseURI, String namespace) **throws** XMLStreamException {

**if** (publicID.equals("bookStore.dtd")) {

**return** Class.**class**.getResourceAsStream("/resources/bookStore.dtd");

}

**return** **null**;

}

});

XMLOutputFactory xof = XMLOutputFactory.*newInstance*();

// Set namespace repairable. Sometimes it will bring you bug. Use it carefully.

// xof.setProperty(XMLOutputFactory.IS\_REPAIRING\_NAMESPACES, true);

InputStream sourceIn = MyStAX.**class**.getResourceAsStream("resources/bookStore.xml");

OutputStream targetOut = System.***out***; //new FileOutputStream(new File("target.xml"));

XMLStreamReader reader = xif.createXMLStreamReader(sourceIn);

XMLStreamWriter writer = xof.createXMLStreamWriter(targetOut, reader.getEncoding());

writer.writeStartDocument(reader.getEncoding(), reader.getVersion());

**while** (reader.hasNext()) {

**int** event = reader.next();

**switch** (event) {

**case** XMLStreamConstants.***DTD***:

*out*(reader.getText());

writer.writeCharacters("\n");

writer.writeDTD(reader.getText());

writer.writeCharacters("\n");

**break**;

**case** XMLStreamConstants.***PROCESSING\_INSTRUCTION***:

*out*(reader.getPITarget());

writer.writeCharacters("\n");

writer.writeProcessingInstruction(reader.getPITarget(), reader.getPIData());

**break**;

**case** XMLStreamConstants.***START\_ELEMENT***:

*out*(reader.getName());

NamespaceContext nc = reader.getNamespaceContext();

writer.setNamespaceContext(reader.getNamespaceContext());

writer.setDefaultNamespace(nc.getNamespaceURI(""));

writer.writeStartElement(reader.getPrefix(), reader.getLocalName(), reader.getNamespaceURI());

**for** (**int** i=0; i<reader.getAttributeCount(); i++) {

QName qname = reader.getAttributeName(i);

String name=qname.getLocalPart();

**if** (qname.getPrefix()!=**null** && !qname.getPrefix().equals("")) {

//name = qname.getPrefix()+":"+name;

}

writer.writeAttribute(name, reader.getAttributeValue(i));

}

**for** (**int** i=0; i<reader.getNamespaceCount(); i++) {

writer.writeNamespace(reader.getNamespacePrefix(i), reader.getNamespaceURI(i));

}

**break**;

**case** XMLStreamConstants.***ATTRIBUTE***:

*out*(reader.getText());

**break**;

**case** XMLStreamConstants.***SPACE***:

*out*("SPACE");

writer.writeCharacters("\n");

**break**;

**case** XMLStreamConstants.***CHARACTERS***:

*out*(reader.getText());

writer.writeCharacters(reader.getText());

**break**;

**case** XMLStreamConstants.***END\_ELEMENT***:

*out*(reader.getName());

writer.writeEndElement();

**break**;

**case** XMLStreamConstants.***END\_DOCUMENT***:

writer.writeEndDocument();

**break**;

**default**:

*out*("other");

**break**;

}

}

writer.close();

reader.close();

}

**public** **static** **void** eventAPIReadWrite() **throws** Exception {

XMLInputFactory xif = XMLInputFactory.*newInstance*();

xif.setProperty(XMLInputFactory.***IS\_NAMESPACE\_AWARE***, **true**);

// Handle the external Entity.

xif.setXMLResolver(**new** MyXMLResolver());

XMLOutputFactory xof = XMLOutputFactory.*newInstance*();

InputStream sourceIn = MyStAX.**class**.getResourceAsStream("resources/bookStore.xml");

OutputStream targetOut = System.***out***;

XMLEventReader reader = xif.createXMLEventReader(sourceIn);

XMLEventWriter writer = xof.createXMLEventWriter(targetOut);

**while**(reader.hasNext()) {

XMLEvent event = reader.nextEvent();

*out*(event.getEventType());

writer.add(event);

}

reader.close();

writer.close();

*out*(" end ");

}

**public** **static** **void** out(Object o) {

System.***out***.println(o);

}

}

**XSLT实例**

上面了解了SAX,DOM和STAX，它们均为XML解析方法. 其中SAX只适合解析读取. DOM则是XML内存中的数据展现. STAX可以解析，也可以写出到文件系统.

如果将DOM从内存输出XML文件. 如果需要将一个XML文件转换成一个HTML或任意其他格式文件，则需要JAXP的XSLT特性. 这里的转换包括：

1. 两个结构不同的DOM相互转换. DOMSouce -----> DOMResult
2. DOM输出到XML.  DOMSource -----> StreamResult
3. DOM转化成另一种格式文件,比如HTML.  DOMSource ---(XSL)--->StreamResult.
4. XML文件转换成另一种格式文件.  SAXSource|StreamSource ---(XSL)---->StreamResult
5. XML文件到DOM. SAXSource|StreamSouce ------> DOMResult
6. DOM到另一个SAX事件 DOMSource------>SAXResult

XSLT的下面包含了4个包：

1. javax.xml.transform - 定义了Transformer类，调用Transformer的transform(source, result)方法，可以进行XML的转换.
2. javax.xml.transform.sax - 里面定义了SAXSource和SAXResult.
3. javax.xml.transfrom.dom - 定义了DOMSource和DOMResult.
4. javax.xml.transform.stream - 定义了StreamSource和StreamResult.
5. javax.xml.transform.stax - 定义了StAXSource和StAXResult.(java1.6)

从上面可以看出，JAXP可以进行4\*4=16种转换方式.(sax, sax), (sax, dom), (sax, stream)...

再高级一点,利用 SAXSouce----->DOMResult的转化功能, 和SAX模拟XML读取功能, XSLT可以将一个非XML格式的文件,转换成一个DOM. 下面的代码将包含此例. 代码中还包含另外一个例子,就是把XML按照XSL的格式转换成HTML.

**注意, XSLT处理DTD有技巧:**  
在xml2html的转换中, 使用StreamSource在代码的书写上是最简单的, 但为 什么使用了SAXSource? 那是因为要转换的XML中引用了DTD, StreamSource无法处理外部引用, 会导致Transformer抛TransformerException. 失败的异常内容为DTD文件找不到. 那么,在这种情况下,我们只能使用SAXSource,并给它赋予一个可以解析外部DTD引用的XMLReader. 终于成功了.

**package** com.train.jaxp;

**import** java.io.IOException;

**import** javax.xml.parsers.SAXParser;

**import** javax.xml.parsers.SAXParserFactory;

**import** javax.xml.transform.Result;

**import** javax.xml.transform.Source;

**import** javax.xml.transform.Transformer;

**import** javax.xml.transform.TransformerFactory;

**import** javax.xml.transform.sax.SAXSource;

**import** javax.xml.transform.stream.StreamResult;

**import** org.xml.sax.ContentHandler;

**import** org.xml.sax.DTDHandler;

**import** org.xml.sax.EntityResolver;

**import** org.xml.sax.ErrorHandler;

**import** org.xml.sax.InputSource;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.SAXNotRecognizedException;

**import** org.xml.sax.SAXNotSupportedException;

**import** org.xml.sax.XMLReader;

**import** com.train.jaxp.handle.MyEntityResolver;

**public** **class** MyXSLT {

TransformerFactory tff;

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

MyXSLT xslt = **new** MyXSLT();

xslt.xml2html();

xslt.str2xml();

}

**public** MyXSLT() {

tff = TransformerFactory.*newInstance*();

}

**public** **void** xml2html() **throws** Exception {

Transformer tr = tff.newTransformer(**new** SAXSource(**new** InputSource(**this**.getClass().getResourceAsStream("resources/bookStore.xsl"))));

SAXParserFactory spf = SAXParserFactory.*newInstance*();

SAXParser parser = spf.newSAXParser();

parser.getXMLReader().setEntityResolver(**new** MyEntityResolver());

Source source = **new** SAXSource(parser.getXMLReader(), **new** InputSource(**this**.getClass().getResourceAsStream("resources/bookStore.xml")));

Result target = **new** StreamResult(System.***out***);

tr.transform(source, target);

}

// "[joey,bill,cat]" will be transformed to

// <test><name>joey</name><name>bill</name><name>cat</name></test>

**public** **void** str2xml() **throws** Exception {

**final** String[] names = **new** String[]{"joey","bill","cat"};

Transformer tr = tff.newTransformer();

Source source = **new** SAXSource(**new** XMLReader() {

**private** ContentHandler handler;

@Override

**public** **void** parse(InputSource input) **throws** IOException,

SAXException {

handler.startDocument();

handler.startElement("", "test", "test", **null**);

**for** (**int** i=0; i<names.length; i++) {

handler.startElement("", "name", "name", **null**);

handler.characters(names[i].toCharArray(), 0, names[i].length());

handler.endElement("", "name", "name");

}

handler.endElement("", "test", "test");

handler.endDocument();

}

@Override

**public** **void** parse(String systemId) **throws** IOException, SAXException {

}

@Override

**public** **boolean** getFeature(String name)

**throws** SAXNotRecognizedException, SAXNotSupportedException {

**return** **false**;

}

@Override

**public** **void** setFeature(String name, **boolean** value)

**throws** SAXNotRecognizedException, SAXNotSupportedException {

}

@Override

**public** Object getProperty(String name)

**throws** SAXNotRecognizedException, SAXNotSupportedException {

**return** **null**;

}

@Override

**public** **void** setProperty(String name, Object value)

**throws** SAXNotRecognizedException, SAXNotSupportedException {

}

@Override

**public** **void** setEntityResolver(EntityResolver resolver) {

}

@Override

**public** EntityResolver getEntityResolver() {

**return** **null**;

}

@Override

**public** **void** setDTDHandler(DTDHandler handler) {

}

@Override

**public** DTDHandler getDTDHandler() {

**return** **null**;

}

@Override

**public** **void** setContentHandler(ContentHandler handler) {

**this**.handler = handler;

}

@Override

**public** ContentHandler getContentHandler() {

**return** handler;

}

@Override

**public** **void** setErrorHandler(ErrorHandler handler) {

}

@Override

**public** ErrorHandler getErrorHandler() {

**return** **null**;

}

}, **new** InputSource());

Result target = **new** StreamResult(System.***out***);

tr.transform(source, target);

}

}