**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

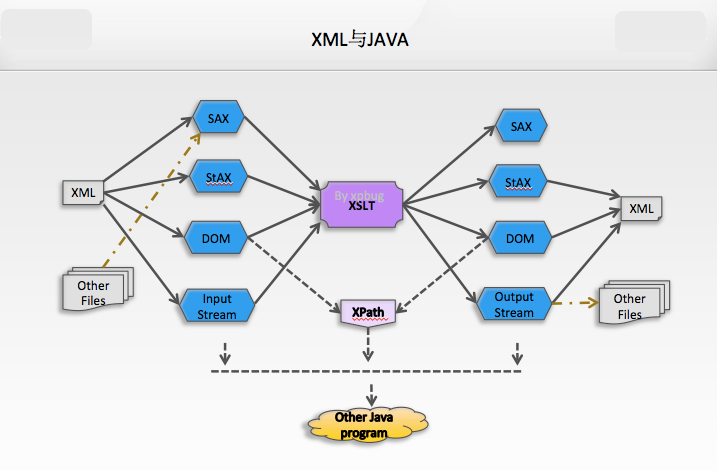
Java对XML的支持被称为JAXP(Java API for XML Processing). JAXP被当做标准,放入了J2SE1.4.从此以后,JRE自带XML的处理类库. 当然,JAXP允许使用第三方的XML Parser,不同的parser有着不同的优缺点,用户可以自己选择. 但所有的Parser均必须实现JAXP所约定的Interface. **掌握JAXP,需要知道以下内容. 这些都会在后面进行描述.**

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.

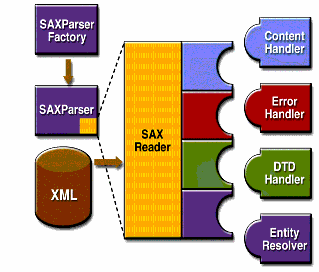
Pull VS Push

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);

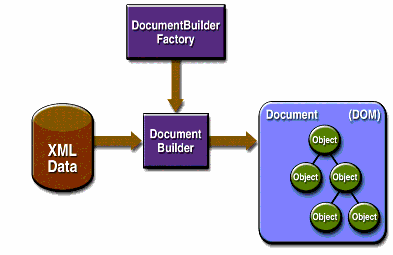
        is.setEncoding("UTF-8");

        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);

}

}

Reference

<http://www.ibm.com/developerworks/cn/xml/x-jaxp/>

http://www.ibm.com/developerworks/cn/xml/x-jaxp2/

# JAXP 全部内容，第 2 部分

使用 Sun 的 XML 处理工具包来转换 XML

在这个由两部分组成的系列文章中，第 1 部分介绍了 Java™ API for XML Processing (JAXP) 及其解析和验证特性。JAXP 还为 Java 程序员提供了使用可扩展样式表语言 (XSL) 转换 XML 文档的能力。通过直接编程访问和 XSL 模板，JAXP 使得从一种 XML 格式转换为另一种格式非常容易。本文展示如何使用 JAXP 来转换 XML 文档和如何高速缓存 XSL 样式表以实现最佳性能。

0http://dw1.s81c.com/developerworks/i/v17/dw-cmts-arrow.png [评论：](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#icomments)

[Brett McLaughlin](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#authorN10028) ([brett@newInstance.com](mailto:brett@newInstance.com?subject=JAXP%20%E5%85%A8%E9%83%A8%E5%86%85%E5%AE%B9%EF%BC%8C%E7%AC%AC%202%20%E9%83%A8%E5%88%86&cc=dwxed@us.ibm.com)), 作家和编辑, O'Reilly Media, Inc.

2005 年 6 月 13 日

* [+](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#toggle)内容



在 IBM Bluemix 云平台上开发并部署您的下一个应用。

[开始您的试用](https://developer.ibm.com/sso/bmregistration?lang=zh_CN&ca=dwchina-_-bluemix-_-x-jaxp2-_-sidebar)

在 JAXP 的早期版本中，该首字母缩写代表 Java API for XML Parsing。在 [第 1 部分](http://www.ibm.com/developerworks/cn/xml/x-jaxp/) 中了解到，JAXP 是位于 SAX 和 DOM 之上的层，它允许 Java 程序员执行开发商中立的 XML 解析。最初，这是 JAXP 的全部特性。不过俗话说的好，过去是过去，现在是现在。

过 去，Java 和 XML 组合本身主要用于解析。Java 应用程序只需读入 XML 文档，然后按程序处理文档的数据。但随着 XML 消费应用程序流行起来，很显然，各种应用程序所执行的操作有许多重叠。对于所有优秀的软件，重叠将导致规范（而且每次都产生新的有用的 API）。

XML 的广泛使用所产生的第一个规范是 XSL（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)）。 应用程序不断提取 XML 数据，添加一些格式化，然后将其显示在用户界面上——通常是作为 HTML、XHTML 或 WML。XSL 完成此任务，并在其上构建规范，从而允许应用程序抛弃其所有的专用转换代码。XSL 规范产生之后，XML 转换 API（Transformation API for XML，TrAX）随之出现（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)）。 TrAX 提供了在 Java 应用程序中使用 XSL 的简单一致的方法。目前，JAXP——相当长的链（和介绍）中的最后一链——已经将 TrAX 合并到核心 Java 开发环境中。按照所有的发展，以及最近的添加（比如扩展验证和 XPath 支持），JAXP 现在代表 Java API for XML Processing。本文重点介绍使用 JAXP 进行处理而非解析。

## 由此及彼

理解 XSL 的基本程序流对于掌握 JAXP 如何处理转换是非常关键的。如果对 XSL 十分陌生，则需要快速回顾一下 XSL 基本概念。即使您是 XSL 专家也要忍受我如此絮叨。

### 来源 (XML)

使用 XSL 时，必须从 XML 开始。我知道这听起来理所当然，但还是值得说明一下。您可能习惯以 XML 文件开始（比如 phonebook.xml），并将其传递到 XSL 处理器中。JAXP 不仅允许您传递文件，它还允许您做好多事，在下一节 [输入和输出](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#inout) 中将学习相关内容。

### 样式表 (XSL)

可能吸引大多数设计人员的是 XSL 样式表。样式表是一个指令集合，它指定特定类型的数据作为输入，并指定其他一组数据和格式化作为输出。但是切记，样式表应该对入站 XML 的结构进行操作，而不是对文档中的特定数据进行操作。这就确保样式表处理任何给定格式的 XML，而非特定的实例文档。

### 目标 (\*ML)

最后需要记住，只能从 XSL 中输出格式良好的标记语言。不能输出 Microsoft Word 文档或 PDF。一定要使用标记语言，比如 XML、XHTML、WML 或其他良好的 \*ML（标记语言）变种。

当 然，对此我听到异议——已经看到了从 XML 输出 PDF 的应用程序，或将 XML 转换为 Excel 的应用程序。而且，可以接受特定格式的 XML 并将其转换为二进制格式的引擎确实存在。 但这并不属于 XSL 的领域；它是转换后处理。JAXP 会帮助转换 XML，但它不允许转换为二进制格式。

[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 输入和输出

通过简单的回顾可能已经了解到，许多 XML 转换只是关于输入和输出的。导入 XML，对它进行操作，然后输出 \*ML。在处理所有中间位（我意识到这是最有趣的地方）之前，将展示如何将数据输入到 JAXP 和如何将其输出返回。

### JAXP 的灵活性

JAXP 不是一般的转换引擎。它不能将 Java 属性文件或一次性文本格式转换为 XML 或其他格式的标记语言。事实上，JAXP 甚至不接受 HTML，除非它是格式良好的（因此产生了一些 XHTML）。所以在试图使 JAXP 成为更一般的 API 之前，应该意识到必须使用格式良好的 XML，否则其他一切都没用。

JAXP 的灵活性在于可以如何表示 XML。显而易见的原因是，JAXP 可以将 XML 接受为文件或包装文件的流。但它还可以将输入接受为 DOM Document（这 表示不可能存在于磁盘上的 XML 文档），或者接受为一系列 SAX 事件（这也表示 XML 文档）。使用这个附加的灵活性，可以将 JAXP 插入到任何 XML 事件链中。例如，如果具有一段代码，该代码使用 SAX 读入 XML 文档，然后将该数据的特定部分传递给另一个应用程序，则可以简单地在 SAX 代码和其他应用程序组件之间插入 JAXP。它可以消费 SAX 事件，按需转换数据，并将结果传递给接收应用程序组件。

这个故事的寓意是，可以以任何标准格式将 XML 传递给 JAXP，并以同样多的格式将其输出。即使目前不需要这种灵活性，或无法想像怎么可能使用所有这些不同格式，但是当您的需要变得更高级时，JAXP 已经准备好为您服务了。

### 输入来源

## 接口编程 101

如果使用接口对您来说相当陌生，则要注意，[清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code1) 总是把接口放在等号 (=) 左边，把特定实现类放在右边。所以 Source 在左边，StreamSource、SAXSource 等实现类在右边。

javax.xml.transform.Source 接口是 JAXP 的所有输入和转换 API 的基础。该接口只定义了两个方法——getSystemId() 和 setSystemId(String systemId)。实际上，您将不会像处理 JAXP 提供的具体实现那样，过多地直接处理该接口：

* **javax.xml.transform.dom.DOMSource** 将 DOM Node（及其孩子）传递给 JAXP。
* **javax.xml.transform.sax.SAXSource** 将 SAX 回调结果（来自 XMLReader）传递给 JAXP。
* **javax.xml.transform.stream.StreamSource** 将包装在 File、InputStream 或 Reader 中的 XML 传递给 JAXP。

清单 1 展示了几种用于创建转换中使用的 Source 的方法：

##### 清单 1. 使用 Source 接口的实现

// Create a Source from a file on disk

Source fileSource =

new StreamSource(new File("phonebook.xml"));

// Create a Source from a DOM tree

Document myDomDocument = getDocument();

Source domSource = new DOMSource(myDomDocument);

// Create a Source from an InputStream

BufferedInputStream bis =

new BufferedInputStream(getInputStream());

Source streamSource = new StreamSource(bis);

// Create a Source from a reader and SAX InputSource

XMLReader myXMLReader = getXMLReader();

InputSource myInputSource = getInputSource();

Source saxSource = new SAXSource(myXMLReader, myInputSource);

[清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code1) 几乎是自解释的。一旦获得 Source，就可以将 XML 输入 JAXP 的 XSL 处理部分。

### 输出结果

在讲述转换本身之前，将简单介绍一下 Source 的输出对应物——javax.xml.transform.Result。它甚至具有与 Source 相同的两个基本方法——getSystemId() 和 setSystemId(String systemId)。

与其输入对应物一样，一般使用 JAXP 的具体 Result 实现：

* **javax.xml.transform.dom.DOMResult** 将转换后的内容传递到 DOM Node 中。
* **javax.xml.transform.sax.SAXResult** 将转换的结果传递到 SAX ContentHandler 中。
* **javax.xml.transform.stream.StreamResult** 将转换后的 \*ML 传递到 File、OutputStream 或 Writer 中。

清单 2 展示了一些简单的例子，与 [清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code1) 中 Source 的例子十分相似：

##### 清单 2. 使用 Result 接口的实现

// Write to a file on disk

Result fileResult =

new StreamResult(new File("output.xml"));

// Write a Result to a DOM tree (inserted into the supplied Document)

Document myDomDocument = getDocument();

Result domResult = new DOMResult(myDomDocument);

// Create a Result from an OutputStream

BufferedOutputStream bos =

new BufferedOutputStream(getOutputStream());

Result streamResult = new StreamResult(bos);

// Create a Result to write to a SAX ContentHandler

ContentHandler myContentHandler = new MyContentHandler();

Result saxResult = new SAXResult(myContentHandler);

一旦理解了 Source 和 Result 接口以及与 JAXP 绑定在一起的实现之后，就差不多已经掌握了 XML 转换。

[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 使用 JAXP 执行转换

如果阅读 [第 1 部分](http://www.ibm.com/developerworks/xml/library/x-jaxp/?S_TACT=105AGX52&S_CMP=cn-a-x) 距今有一段时间了，或者谈到 JAXP 和解析时仍有些生疏，就应该花时间回顾一下 SAXParserFactory 和 DOMBuilderFactory 类。您将发现，如果知道如何使用这些类，就已经能够完全理解 JAXP 转换如何工作了。

### 获得工厂

转换如此简单，以至于有点微不足道。首先，需要设置输入和输出接收器。将 Source 包装在输入 XML 文档和 XSL 样式表中。然后，创建一个接收器，以写入转换后的结果——然后将其包装在 Result 中。

其次，需要使用静态 newInstance() 方法创建 TransformerFactory。清单 3 展示了所有详细信息：

##### 清单 3. 创建新 TransformerFactory 实例

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

这一步没有太多内容。异常处理与代码本身所用时间一样多。对于 SAX 和 DOM 工厂类，一个异常处理已请求的但不受支持的特性，另一个异常处理实例化错误。

## 恒等转换

TransformerFactory.newTransformer() 的一个版本不接受任何参数（因此无 XSL 样式表）。这允许您执行**恒等转换**，它简单地将输入 XML 从一种形式（比如流）转换为另一种形式（比如 DOM 树）。您以一种格式提供 XML 作为 Source，然后以另一种格式将其推出作为 Result。应该记住这个有用的技巧。

工厂类本身用于获得 Transformer 的实例（在 [下一小节](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#transfrm) 中讨论），并执行简单配置。可以使用 setFeature(String feature, boolean value) 方法来调用处理器上的特性。当然，工厂上设置的任何特性都应用于由此工厂创建的所有 Transformer 实例。

### 创建 Transformer

下一步是获得对象来执行实际转换。这是另一段相当令人厌烦的代码：只在工厂上调用 newTransformer()，并为该方法提供要使用的 XSL 样式表。清单 4 展示了详细操作：

##### 清单 4. 使用 TransformerFactory 创建 Transformer

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

**// Get a transformer for this XSL**

**Transformer transformer = factory.newTransformer(inputXSL);**

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

此处没有太多值得注意的地方；惟一需要确保具有的就是 Transformer 与特定样式表之间的连接。因为样式表用于创建 Transformer，这是惟一可以用于该实例的 XSL。如果想要使用不同的样式表执行附加转换，可以重用 TransformerFactory，但必须创建不同的 Transformer 实例，以与新样式表连接。

### 执行转换

一切就绪之后，只需要一行代码来执行转换。清单 5 展示了如何使用 transform() 方法。只需为它提供输入 XML 和输出接收器；样式表已经连接到要使用的 Transformer 实例上：

##### 清单 5. 使用 transform() 方法

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

// Get a transformer for this XSL

Transformer transformer = factory.newTransformer(inputXSL);

**// Perform the transformation**

**transformer.transform(inputXML, outputXHTML);**

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

调用该方法之后，转换的结果写出到所提供的 Result 中。在 [清单 5](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code5) 中，这是一个文件，但还可以将输出发送到 SAX ContentHandler 或 DOM Node 中。如果想要全部尝试，绑定的文件提供了简单的 XML 文件、XSL 样式表和源代码（参阅 [下载](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#download)）。

[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 高速缓存 XSL 样式表

这一切都很简单，但这样使用 JAXP 有两个明显的限制：

* 每次执行 transform()，Transformer 对象都要处理 XSL 样式表。
* Transformer 的实例不是线程安全的。在多个线程之间不能使用相同的实例。

这两个限制源自同一问题：Transformer 每次执行转换时，都必须重新处理 XSL。如果该处理出现在多个线程中，就可能发生严重问题。在线程问题之上，必须一再地付出处理 XSL 样式表的成本。毫无疑问，您非常希望知道如何解决这些问题，那就继续读下去。

### 加载模板

尚未讨论过的接口 javax.xml.transform.Templates 与 javax.xml.transform.Transformer 相近。Templates 接口是线程安全的（解决了第二个限制），并代表**已编译**的样式表（解决第一个限制）。在讨论相关概念之前，请查看清单 6：

##### 清单 6. 使用 JAXP Templates 接口

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output-templates.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

**// Pre-compile instructions**

**Templates templates = factory.newTemplates(inputXSL);**

**// Get a transformer for this XSL**

**Transformer transformer = templates.newTransformer();**

// Perform the transformation

transformer.transform(inputXML, outputXHTML);

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

[清单 6](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code6) 中的黑体行表示需要从 [清单 5](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code5) 进行的惟一更改。不是使用工厂来直接获得 Transformer，而是使用 newTemplates() 方法；这将返回一个 Templates 对象，它是线程安全的。可以将该对象传递到其他线程中的其他方法，而且根本无需担心。因为它将从传递它的 XSL 中预编译转换指令，所以传递给其他方法甚至线程是安全的。

然后，从 Templates.newTransformer() 方法中获得 Transformer 实例。在这个阶段无需指定 XSL，因为 Transformer 已经处理过了（事实上，它是已编译的 XSL，所以可以不更改样式表，如果您愿意的话）。除了多出的一行以及对现有行的一个更改之外，再无其他新内容。相当酷，考虑一下您的代码因为这个小小的更改变得有多好。

### 从 Transformer 到 Templates

最后一个值得考虑的问题是，何时使用从工厂直接获得的 Transformer 和何时使用 Templates 对象。我几乎总是选择使用 Templates 对象，因为使用 XSL 时，我总是重复使用相同的样式表。与其花时间在 XSL 上进行多次传递，我宁愿将指令预编译到 Templates 对象中，完成 XSL 处理。

也就是说，在一些情况下，最好是直接从 TransformerFactory 中拖出 Transformer。如果您知道您将只使用特定样式表执行单个转换，那么不预编译到 Templates 对象中会比较快，因为预编译需要略多一点的开销。但是，需要确保没有重用。在我的（完全不科学，使用了简短的样例）测试中，我发现如果使用一个 XSL 样式表两次，使用 Templates 对象和直接使用 Transformer 不分上下。 一旦使用三次以上，Templates 方法就要好多了。您还需要确保将没有任何线程技术问题；但这是一件简单的事情，所以我把它留给您应用在编程中。 一般地，使用 Templates 对象通常更安全。

[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 更改 XSL 处理器

在 [第 1 部分](http://www.ibm.com/developerworks/xml/library/x-jaxp/?S_TACT=105AGX52&S_CMP=cn-a-x) 中已经看到，通过更改系统属性，可以用自己的实现替换默认 JAXP 解析器实现。同一原则适用于 XSL 处理器。JAXP 预包装有 Xalan-J（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)），我总是使用它。但灵活性总是好事，而 JAXP 提供了这一点。

如果想要使用除 Xalan 之外的处理器，请为名为 javax.xml.transform.TransformerFactory 的系统属性提供一个值。需要为该属性分配要实例化的类的名称。该类应继承 javax.xml.transform.TransformerFactory（当然，这也是要设置的系统属性的名称），并填充方法其余的抽象。仅使用如下代码：

java -Djavax.xml.transform.TransformerFactory

=[transformer.impl.class] TestTransformations

simple.xml simple.xsl

全部结束！

[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 结束语

在早期版本中，JAXP 不过是位于 SAX 和 DOM 以及那些 API 过时版本之上的小薄层。现在，使用 JAXP 1.3 可以解析、验证并转换 XML，而无需编写一行特定于开发商的代码。虽然追溯到 SAX 代码、使用类似 DTDParser（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)）的工具或甚至自己处理转换，这些都是有意义的，但在 API 和工具的仓库中您需要 JAXP。甚至可能比开发商中立性更重要的一点是，具有最新的 Java 虚拟机 (JVM) 的所有客户和客户机都将具有 JAXP。所以使用它，好好使用它，经常使用它。

[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 下载

| **描述** | **名字** | **大小** |
| --- | --- | --- |
| **Sample code for All about JAXP, Part 2** | [x-jaxp2-all-about.zip](http://www.ibm.com/developerworks/apps/download/index.jsp?contentid=162427&filename=x-jaxp2-all-about.zip&method=http&locale=zh_CN) | 4 KB |

## 参考资料

* 您可以参阅本文在 developerWorks 全球站点上的 [英文原文](http://www.ibm.com/developerworks/xml/library/x-jaxp2/?S_TACT=105AGX52&S_CMP=cn-a-x)。
* 请阅读这个两部分系列的 [JAXP 全面介绍，第 1 部分](http://www.ibm.com/developerworks/cn/xml/x-jaxp/)，其中介绍了 JAXP 及其解析和验证特性。
* 登录由 Brett McLaughlin 主办的“[XML and Java technology](http://www.ibm.com/developerworks/forums/dw_forum.jsp?S_TACT=105AGX52&cat=11&S_CMP=cn-a-x&forum=262)”论坛，获得有关如何使用这些技术的其他信息。
* 在 Sun 的 [Java and XML headquarters](http://java.sun.com/xml/) 上学习有关 JAXP 的更多知识。
* 如果对 Java 编程完全陌生，可以通过下载 [Java 5.0](http://java.sun.com/j2se/1.5.0/download.jsp) 获得 JAXP 和完整的 JDK。
* 从万维网联盟 (W3C) 中获得 [XSL](http://www.w3.org/Style/XSL/) 标准的完整信息。
* 下载 [Apache Xalan](http://xml.apache.org/xalan-j/)，这个 XSL 处理器在 Sun 的 JDK 5.0 实现中。
* Sun 使用它的 JDK 5.0 实现中的 [Apache Xerces](http://xml.apache.org/xerces2-j/) 解析器。
* 要深入了解 JAXP 1.3 中的新特性，请阅读 developerWorks 的两部分系列“What's new in JAXP 1.3?”[JAXP 1.3 的新特性，第 1 部分](http://www.ibm.com/developerworks/cn/xml/x-jaxp13a.html) （2004 年 11 月），其中提供了 JAXP 规范的概述，给出了 javax.xml.parsers 软件包修订的详细信息，并描述了功能强大的模式高速缓存和验证框架。 [JAXP 1.3 的新特性，第 2 部分](http://www.ibm.com/developerworks/cn/xml/x-jaxp13b.html) （2004 年 12 月）介绍了一些实用程序，这些程序添加了对在 XML 规范中名称空间中定义的概念的支持，并描述了对 javax.xml.transform 软件包的更改。
* 阅读 [Transformation API for XML (TrAX)](http://xml.apache.org/xalan-j/trax.html) 的详细信息，TrAX 现在是 JAXP 的一部分。
* 阅读“[Putting XSL transformations to work](http://www.ibm.com/developerworks/xml/library/x-xsltwork/index.html?S_TACT=105AGX52&S_CMP=cn-a-x)”（developerWorks，2001 年 10 月）中 XSL 的介绍，以及从使用 XSL 转换而获利的现实世界商业事例的讨论。
* 阅读 Brett McLaughlin 的书籍 [Java & XML](http://devworks.krcinfo.com/WebForms/ProductDetails.aspx?ProductID=0596001975)（O'Reilly &Associates，2001 年），本书解释了 Java 程序员可以如何使用 XML 构建基于 Web 的企业应用程序。
* 在 JAXP 的介绍中查找有关 API 的详细信息。先从 SAX Web 站点上的 [SAX 2 for Java](http://www.saxproject.org) 开始，然后再查看 W3C Web 站点上的 [DOM](http://w3.org/DOM)。
* 学习有关 [JDOM](http://www.jdom.org/) 的更多知识，这是一个开放源码工具包，提供了用 Java 语言表示 XML 文档的方法，以方便进行简单有效的读取、写入和操纵。
* 阅读“[使用 XML: 链接管理和面对未来](http://www.ibm.com/developerworks/cn/xml/x-wxxm/part3/index.html)”（developerWorks，2001 年 9 月），学习如何在简单的内容管理系统中使用多个样式表。
* 从 Doug Tidwell 的 developerWorks 教程 “[利用 Java 技术进行 XML 编程，第 1 部分](http://www.ibm.com/developerworks/cn/views/xml/tutorials.jsp?cv_doc_id=85087)”（2004 年 1 月），学习使用 Java 技术操纵 XML 文档的基本知识。 [第 2 部分](http://www.ibm.com/developerworks/cn/views/xml/tutorials.jsp?cv_doc_id=85129)（2004 年 7 月）介绍了比较难的主题，比如使用名称空间、验证 XML 文档和无需典型的 XML 文档来构建 XML 结构。 最后，[第 3 部分](http://www.ibm.com/developerworks/cn/views/xml/tutorials.jsp?cv_doc_id=85131)（2004 年 7 月）展示如何完成更高级的任务，比如生成 XML 数据结构、操纵这些结构，以及连接 XML 解析器与非 XML 数据源。
* 如果对直接处理 DTD 而不是通过验证 API 的解析器感兴趣，请查阅 Mark Wutka 便捷的 [DTDParser](http://www.wutka.com/dtdparser.html)。
* 在 developerWorks [Developer Bookstore](http://devworks.krcinfo.com/) 上大致浏览一下与 XML 相关的书名。
* 了解如何才能成为一名 [IBM 认证的 XML 及相关技术的开发人员](http://www.ibm.com/certify/certs/adcdxmlrt.shtml)。

为了让应用程序有效地处理XML数据，你必须向SAX解析器注册处理程序。处理程序也称Handler接口，是由SAX定义的一组回调方法组成的，这些方法使你可以在相关的事件发生时对其进行编程。

在SAX2.0中定义了**四**大核心接口：org.xml.sax.ContentHandler，org.xml.sax.ErrorHandler，org.xml.sax.DTDHandler以及org.xml.sax.EntityResolver。

其于SAX的XML应用程序必须实现一个或多个Handler接口，并为Handler接口中的回调方法编程（如果你不想添加代码，也可以不写，这样就可以忽略这种类型的事件）。然后使用**XMLReader**中的setContentHandler()，setErrorHandler()，setDTDHandler()和setEntityResolver()方法**进行注册**即可。在解析过程中，reader会在合适的处理类中调用这些回调方法。

我们将从实现ContentHandler接口开始讲解，ContentHandler，顾名思义，表示的是与XML文档内容处理有关的事件，如元素，属性，字符数据等。示例：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | package xml;    import java.io.File;  import java.io.FileInputStream;  import org.xml.sax.Attributes;  import org.xml.sax.ContentHandler;  import org.xml.sax.InputSource;  import org.xml.sax.Locator;  import org.xml.sax.SAXException;  import org.xml.sax.XMLReader;  import org.xml.sax.helpers.XMLReaderFactory;    public class ParseXML1 {      public static void main(String[] args) {          try {                      XMLReader reader = XMLReaderFactory.createXMLReader();                      reader.setContentHandler(new MyContentHandler());    //注册                      reader.parse(new InputSource(new FileInputStream(                              new File("/home/fuhd/apk/gw/com.application.zomato.apk/AndroidManifest.xml"))));                  } catch (SAXException e1) {                      e1.printStackTrace();                  } catch(Exception e2){                      e2.printStackTrace();                  }      }  }    //Handler接口实现，这里没有实现什么内容，稍后实现  class MyContentHandler implements ContentHandler {      @Override          public void setDocumentLocator(Locator locator) {              // TODO Auto-generated method stub          }      @Override          public void startDocument() throws SAXException {              // TODO Auto-generated method stub          }      @Override          public void endDocument() throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void startPrefixMapping(String prefix, String uri)                          throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void endPrefixMapping(String prefix) throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void startElement(String uri, String localName, String qName,                          Attributes atts) throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void endElement(String uri, String localName, String qName)                          throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void characters(char[] ch, int start, int length)                          throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void ignorableWhitespace(char[] ch, int start, int length)                          throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void processingInstruction(String target, String data)                          throws SAXException {              // TODO Auto-generated method stub            }      @Override          public void skippedEntity(String name) throws SAXException {              // TODO Auto-generated method stub            }  } |

**文档定位器**

第一个需要实现的回调方法是setDocumentLocator()，它用于设置在其他SAX事件中需要使用的org.xml.sax.Locator对象。当某个回调事件被触发时，实现了SAX解析器处理接口的类通常需要在XML文件中找到相应的位置。Locator类有几个很有用的方法，例如getLineNumber()和getColumnNumber()，它们可以返回调用时正在解析的XML文档的位置。例：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class MyContentHandler implements ContentHandler {      private Locator locator;      @Override      public void setDocumentLocator(Locator locator) {          this.locator = locator;      }      //...........其它回调方法...............  } |

**警告**：Locator实例只能在ContentHandler实现的作用域范围内使用，在解析过程外，使用Locator对象，其结果是难以预料的（也是无用的）。

**文档解析的开始和结束**

任何处理过程都有开始和结束。这两个重要的事件都只会发生一次：前者在所有其他解析事件发生之前发生，而后者则是在所有解析事件之后发生。SAX提供了回调方法：startDocument()和endDocument()来表示这些事件。

startDocument()方法在所有其他解析事件的回调方法之前被调用，这样就保证了解析有一个明确的起始点。endDocument()在所有处理类中总是最后被调用的方法，即使在发生错误从而导致解析被终止的情况下也是如此。**注意**：若有不可恢复的错误发生，则ErrorHandler类的回调方法就会被调用，最后再调用endDocument()方法结束解析过程。示例代码：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Override  public void startDocument() throws SAXException {      // TODO Auto-generated method stub  }  @Override  public void endDocument() throws SAXException {      // TODO Auto-generated method stub   } |

示例代码中，无需对这些方法进行任何处理，但因为实现了ContentHandler接口，因而仍给出方法的实现。

**处理指令**

在XML中处理指令比较特殊。它们并非XML的元素，而是由调用程序来进行不同的处理。由于这些特征，SAX定义了特殊的回调方法来处理指令。processingInstruction()这个方法接收处理指令的目标和发送给处理指令的所有数据。

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Override  public void processingInstruction(String target, String data)                  throws SAXException {      // TODO Auto-generated method stub   } |

**名称空间回调方法**

除XML模式（Schema）外，XML名称空间是自最早的XML 1.0推荐标准以来，XML中增加的最为重要的概念。在SAX2.0中，从元素层面就支持名称空间。这样可以将元素前缀和相关名称空间URI表示的元素名 称空间，和一个元素的本地名区分开来。本地名（Local name）指的是没有前缀的元素名。例如，元素rdf:li，其中li就是本地名，rdf是名称空间的前缀，其名称空间的URI可能被声明为：http://www.w3.org/1999/02/22-rdf-syntax-ns#。

有两个SAX回调方法专门用于处理名称空间。当解析器到达前缀映射（Prefix Mapping）的开头和结尾时，它们将被调用。虽然前缀映射是一个新术语，但并非新概念，它就是一个用xmlns属性来声明名称空间的元素。虽然它通常是在根元素中使用，但在XML文档中的任何元素都能声明明确的名称空间。例如：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <catalog>      <books>          <book title="XML in aNutshell" xmlns:xlink="http://www.w3.org/1999/xlink">              <cover xlink:type="simple" xlink:show="onLoad" xlink:href="xmlnutCover.jpg"                  ALT = "XML in a Nutshell" width="125" height="350" />          </book>      </books>  </catalog> |

在本例中，文档中声明了一个明确的名称空间，内置了几个元素。这样元素和元素中的属性都可以获得前缀和URI映射（分别是xlink和http://www.w3.org/1999/xlink）。

startPrefixMapping()回调方法的参数是名称空间前缀和与其相关联的URI。当声明映射的元素结束时，这时映射被称为是“关闭的”或“结束的”，这将触发endPrefixMapping()方法。回调方法唯一的问题是不能以连续方式操作，而SAX通常都是按照这种方式构建的。前缀映射回调恰在声明名称空间的元素回调之前发生，且映射的结束将恰在声明元素完成后导致一个事件发生。如下例：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | ......  /\*\*将URI保存在前缀映射\*/  private Map namespaceMappings;    public JTreeHandler(DefaultTreeModel treeModel,DefaultMutableTreeNode base) {      this.treeModel＝ treeModel;      this.current = base;      this.namespaceMappings = new HashMap();  }  public void startPrefixMapping(String prefix,String uri) {      namespaceMappings.put(uri,prefix);  }  public void endPrefixMapping(String prefix) {      for(Iterator i = namespaceMappings.keySet().iterator();i.hasNext();){          String uri = (String)i.next();          String thisPrefix = (String) namespaceMappings.get(uri);          if(prefix.equals(thisPrefix)){              namespaceMappings.remove(uri);              break;          }      }  } |

见上例，在startPrefixMaping()中添加所报告的前缀和URI，然后在endPrefixMapping()中删除。如果你不保存通过startPrefixMapping()报告的这些前缀，它们在元素的回调代码中就不会出现。最简单的保存方式是使用Map。

**注意**：我们使用URI而不是前缀作为映射的关键字，因为startElement()方法会报告元素的名称空间URI，而不是前缀，因此以URI作为关键字可以使查询更快。然后，在endPrefixMapping()中可以看到，当映射不可用的时候，在删除映射时要增加一些工作。

**元素回调方法**

与获取数据有关的三个主要事件是：元素的开始，结束和character()回调方法。它们告诉我们，已经解析了元素的开始标签，并且找到了元素包含的数据，以及何时执行到元素结束标签。

其中第一个回调方法startElement()为程序提供XML元素及其属性的信息。这个回调方法的参数为元素的名称（有多种形式）和一个org.xml.sax.Attributes实例。Attribute接口（或者说，解析器对其的实现）保存着元素中所有属性的引用。它可以以类似Vector的形式简单地遍历元素的属性。除了能够按索引引用属性（当遍历所有属性时使用）外，也可以使用名字来 引用属性。当然，当我们看到表示XML元素或属性的“名字”这个词时必须要小心，因为它可以指代很多东西。在这种情况下，我们可以使用属性的QName， 也可以在用到了名称空间时，使用本地名和名称空间URI的组合。还有一些辅助方法可以给属性加上名称空间前缀信息，例如getURI(int index)和getLocalName(int index)。总体上来讲，属性接口为元素属性提供了丰富的信息。

除元素属性之外，我们还获得了多种形式的元素名字。这仍然要考虑到XML名称空间。首先提供的是元素的名称空间URI。这样就把元素放到了文档的名称空间完整集合的合适的上下文中，即前面提及的无前缀元素名字。另外也提供了QName。

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @Override  public void startElement(String uri, String localName, String qName,                  Attributes atts) throws SAXException {      // TODO Auto-generated method stub  }  @Override  public void endElement(String uri, String localName, String qName)                 throws SAXException {      // TODO Auto-generated method stub  } |

**元素数据**

标识好程序中元素块的开始和结束，列举了元素属性后，下一个重要的信息是包括在元素自身之中的实际数据。它通常由另外一些元素，文本数据或是两者结合组成。 当出现其他元素时，将初始化这些元素的回调方法同时进行一种伪递归（pseudorecursion）操作：元素之间的嵌套会导致回调方法之间的“嵌 套”。而在某些时候又会碰到文本数据。通常来说对XML客户程序而言，最重要的信息就是这些文本数据了，因为这些数据要么是需要在客户端显示，要么是对客 户端的响应而生成的数据。不同解析器的实现方式是不一样的，通常会设计算法来提高解析速度。

在SAX中，元素中的文本数据通过characters()回调方法传递给应用程序，这个方法给应用程序提供了字符数组，以及要读取的字符的起始索引和长度。从这个数组中生成一个字符串，并使用这个数据：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4 | public void characters(char[] ch,int start, int length) throws SAXException {      String s = new String(ch, start ,length);      //.............................  } |

**注意**：characters()回调方法接收一个字符数组，以及起始索引和长度作为参数，去标识从哪个位置开始读取数据，以及读到数组的哪个位置为止。这会产生某些混乱，常见的错误就是像这样读取字符数组中的数据：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4 | public void characters(char[] ch,int start,int length) throws SAXException {      for(int i=0; i<ch.length; i++)          System.out.println(ch[i]);  } |

这里的错误在于它是从头到尾的读取字符数组。而不是从指定的起始索引读到指定的长度为止。

另外，characters()方法经常会报告空白，这很容易引起混乱，因为ignorableWhitespace()回调方法也会报告空白。你可以记住一个简单的规则来避免这种混乱：”不使用模式（Schema），就不会调用ignorableWhitespace()方法“。**注意**：模式表示约束模型，包括DTD，XML Schema，Relax等。如果没有DTD与XML Schema，将不会触发ignorableWhitespace()回调方法，只触发characters()回调方法。我们再深入一步进行探讨，如果元素中只能包含其他元素，事情就变得相当清楚了，元素之间的空白是可以被忽略的。但是，考虑如下这个混合的内容模型：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1 | <!ELEMENT LINE            (#PCDATA | STAGEDIR)\*> |

在这个模型中，无论是否关联到DTD或者Schema，LINE的开始标签和结束标签之间没有可以忽略的空白。那是因为它无法区分这些空白是用来增加可读性还是文档本身就含有的。例如：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5 | <SPEAKER>CELIA</SPEAKER>  <LINE>      <STAGEDIR>Reads</STAGEDIR>  </LINE>  <LINE>Why should this a desert be?</LINE> |

在这个XHTML程序段中，位于LINE元素和STAGEDIR元素之间的空白是不能被忽略的，要通过characters()回调方法报告出来，提供给予字符有关的回调方法。

**可忽略的空白**

我们已经深入讨论了空白的相关知识，这样实现ignorableWhitespace()方法显然就不在话下了，因为空白是可被忽略的，所以我们代码直接将它忽略就可以了：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3 | public void ignorableWhitespace(char[] ch,int start ,int length) throws SAXException {      //这是可忽略的，因此不需要显示出来  } |

**实体**

实体经常用来引用其他的XML程序段和特殊字符（例如 & 和  >）。 当XML文档被解析时，引用其他文档的实体通常会被展开，并将其插入到XML文档中。然而，非验证型的解析器不要求解析实体引用，而是会忽略它； 另外，你也可以设置解析器有意地忽略实体。在这些情况下，当实体被忽略时，SAX就会调用一个回调方法来说明这种情况的发生。回调方法将给出实体名字，如 何：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3 | public void skippedEntity(String name) throws SAXException {      //忽略  } |

大多数情况下，你不会看到这个回调方法被执行，目前绝大部分已有的解析器，即使是非验证型的，都不会忽略实体。例如Apache Xerces解析器就不会调用这个回调方法，相反地，实体引用将会被扩展，并将其结果包含在XML数据当中返回给应用程序。注意：传递给回调方法的参数不要包括实体引用中前面的 & 符号和最后的分号。对于**&header;**来说，只需要把实体引用的名字header，传递给skippedEntity()就可以了。

**结果**

最后，我要向已经实例化的**XMLReader**注册内容处理类。这是通过setContentHandler()方法完成的：

[?](http://my.oschina.net/fhd/blog/367402)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public static void main(String[] args) {      try {          XMLReader reader = XMLReaderFactory.createXMLReader();          reader.setContentHandler(new MyContentHandler());    //注册          reader.parse(new InputSource(new FileInputStream(              new File("/home/fuhd/apk/gw/com.application.zomato.apk/AndroidManifest.xml"))));      } catch (SAXException e1) {          e1.printStackTrace();      } catch(Exception e2){          e2.printStackTrace();      }  }  class MyContentHandler implements ContentHandler {//....................} |