Dom4j 使用简介

DOM4J 是 dom4j.org 出品的一个开源 XML 解析包，它的网站中这样定义：

Dom4j is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP.

Dom4j 是一个易用的、开源的库，用于 XML，XPath 和 XSLT。它应用于 Java 平台，采用了 Java 集合框架并完全支持 DOM，SAX 和 JAXP。

DOM4J 使用起来非常简单。只要你了解基本的 XML-DOM 模型，就能使用。然而他自己带的指南只有短短一页（html），不过说的到挺全。国内的中文资料很少。因而俺写这个短小的教程方便大家使用，这篇文章仅谈及基本的用法，如需深入的使用，请……自己摸索或查找别的资料。

之前看过 IBM developer 社区的文章（参见附录），提到一些 XML 解析包的性能比较，其中 DOM4J 的性能非常出色，在多项测试中名列前茅。（事实上 DOM4J 的官方文档中也引用了这个比较）所以这次的项目中我采用了 DOM4J 作为 XML 解析工具。

在国内比较流行的是使用 JDOM 作为解析器，两者各擅其长，但 DOM4J 最大的特色是使用大量的接口，这也是它被认为比 JDOM 灵活的主要原因。大师不是说过么，“面向接口编程”。目前使用 DOM4J 的已经越来越多。如果你善于使用 JDOM，不妨继续用下去，只看看本篇文章作为了解与比较，如果你正要采用一种解析器，不如就用 DOM4J 吧。

它的主要接口都在 org.dom4j 这个包里定义：

Attribute	**Attribute** 定义了 **XML** 的属性
Branch	**Branch** 为能够包含子节点的节点如 **XML** 元素 **(Element)**和文档**(Docuemnts)**定义了一个公共的行为，
CDATA	**CDATA** 定义了 **XML CDATA** 区域
CharacterData	**CharacterData** 是一个标识借口，标识基于字符的节点。如 **CDATA**，**Comment, Text.**
Comment	**Comment** 定义了 **XML** 注释的行为
Document	定义了 **XML** 文档
DocumentType	**DocumentType** 定义 **XML DOCTYPE** 声明
Element	**Element** 定义 **XML** 元素
ElementHandler	**ElementHandler** 定义了 **Element** 对象的处理器
ElementPath	被 **ElementHandler** 使用，用于取得当前正在处理的路径层次信息
Entity	**Entity** 定义 **XML entity**
Node	**Node** 为所有的 **dom4j** 中 **XML** 节点定义了多态行为
NodeFilter	**NodeFilter** 定义了在 **dom4j** 节点中产生的一个滤镜或谓词的行为（**predicate**）
ProcessingInstruction	**ProcessingInstruction** 定义 **XML** 处理指令**.**
Text	**Text** 定义 **XML** 文本节点**.**
Visitor	**Visitor** 用于实现 **Visitor** 模式**.**
XPath	**XPath** 在分析一个字符串后会提供一个 **XPath** 表达式

看名字大致就知道它们的涵义如何了。

要想弄懂这套接口，关键的是要明白接口的继承关系：

- interface java.lang.Cloneable
  -
  -
  - interface org.dom4j.Node
    - interface org.dom4j.Attribute
    -
    - interface org.dom4j.Branch
      - interface org.dom4j.Document
      -
      - interface org.dom4j.Element
    - interface org.dom4j.CharacterData
      -
      -
      - interface org.dom4j.CDATA
      -
      - interface org.dom4j.Comment
      -
      - interface org.dom4j.Text
    -
    - interface org.dom4j.DocumentType
    -
    - interface org.dom4j.Entity
    -
    - interface org.dom4j.ProcessingInstruction

一目了然，很多事情都清楚了。大部分都是由 Node 继承来的。知道这些关系，将来写程序就不会出现 ClassCastException 了。

下面给出一些例子（部分摘自 DOM4J 自带的文档），简单说一下如何使用。

1．　　读取并解析 XML 文档：

读写 XML 文档主要依赖于 org.dom4j.io 包，其中提供 DOMReader 和 SAXReader 两类不同方式，而调用方式是一样的。这就是依靠接口的好处。

```
// 从文件读取 XML，输入文件名，返回 XML 文档
public Document read(String fileName) throws
MalformedURLException, DocumentException {
    SAXReader reader = new SAXReader();
    Document document = reader.read(new File(fileName));
    return document;
}
```

其中，reader 的 read 方法是重载的，可以从 InputStream, File, Url 等多种不同的源来读取。得到的 Document 对象就带表了整个 XML。

根据本人自己的经验，读取的字符编码是按照 XML 文件头定义的编码来转换。如果遇到乱码问题，注意要把各处的编码名称保持一致即可。

2．　　取得 Root 节点

读取后的第二步，就是得到 Root 节点。熟悉 XML 的人都知道，一切 XML 分析都是从 Root 元素开始的。

```
public Element getRootElement(Document doc){
  return doc.getRootElement();
}
```

3． 遍历 XML 树

DOM4J 提供至少 3 种遍历节点的方法：

1) 枚举(Iterator)

```
// 枚举所有子节点
for ( Iterator i = root.elementIterator(); i.hasNext(); ) {
  Element element = (Element) i.next();
  // do something
}
// 枚举名称为 foo 的节点
for ( Iterator i = root.elementIterator(foo); i.hasNext();) {
  Element foo = (Element) i.next();
  // do something
}
// 枚举属性
for ( Iterator i = root.attributeIterator(); i.hasNext(); ) {
  Attribute attribute = (Attribute) i.next();
  // do something
}
```

2)递归

递归也可以采用 Iterator 作为枚举手段，但文档中提供了另外的做法

```
public void treeWalk() {
  treeWalk(getRootElement());
}
public void treeWalk(Element element) {
```

```
    for (int i = 0, size = element.nodeCount(); i < size; i++)    {
       Node node = element.node(i);
       if (node instanceof Element) {
         treeWalk((Element) node);
       } else { // do something....
       }
    }
}
```

3) Visitor 模式

最令人兴奋的是 DOM4J 对 Visitor 的支持，这样可以大大缩减代码量，并且清楚易懂。了解设计模式的人都知道，Visitor 是 GOF 设计模式之一。其主要原理就是两种类互相保有对方的引用，并且一种作为 Visitor 去访问许多 Visitable。我们来看 DOM4J 中的 Visitor 模式(快速文档中没有提供)

只需要自定一个类实现 Visitor 接口即可。

```
      public class MyVisitor extends VisitorSupport {
       public void visit(Element element){
          System.out.println(element.getName());
       }
       public void visit(Attribute attr){
          System.out.println(attr.getName());
       }
    }

    调用：  root.accept(new MyVisitor())
```

Visitor 接口提供多种 Visit()的重载，根据 XML 不同的对象，将采用不同的方式来访问。上面是给出的 Element 和 Attribute 的简单实现，一般比较常用的就是这两个。VisitorSupport 是 DOM4J 提供的默认适配器，Visitor 接口的 Default Adapter 模式，这个模式给出了各种 visit(*)的空实现，以便简化代码。

注意，这个Visitor是自动遍历所有子节点的。如果是root.accept(MyVisitor)，将遍历子节点。我第一次用的时候，认为是需要自己遍历，便在递归中调用Visitor，结果可想而知。

4．XPath支持

DOM4J对XPath有良好的支持，如访问一个节点，可直接用XPath选择。

```
public void bar(Document document) {
    List list = document.selectNodes( //foo/bar );
    Node node = document.selectSingleNode(//foo/bar/author);
    String name = node.valueOf( @name );
  }
```

例如，如果你想查找XHTML文档中所有的超链接，下面的代码可以实现：

```
  public void findLinks(Document document) throws
DocumentException {
    List list = document.selectNodes( //a/@href );
    for (Iterator iter = list.iterator(); iter.hasNext(); ) {
       Attribute attribute = (Attribute) iter.next();
       String url = attribute.getValue();
    }
  }
```

5．字符串与XML的转换

有时候经常要用到字符串转换为XML或反之，

```
// XML转字符串
Document document = ...;
```

```
    String text = document.asXML();
// 字符串转 XML
    String text = <person> <name>James</name> </person>;
    Document document = DocumentHelper.parseText(text);
```

6 用 XSLT 转换 XML

```
  public Document styleDocument(
      Document document,
      String stylesheet
    ) throws Exception {
    // load the transformer using JAXP
    TransformerFactory factory = TransformerFactory.newInstance();
    Transformer transformer = factory.newTransformer(
      new StreamSource( stylesheet )
    );
    // now lets style the given document
    DocumentSource source = new DocumentSource( document );
    DocumentResult result = new DocumentResult();
    transformer.transform( source, result );
    // return the transformed document
    Document transformedDoc = result.getDocument();
    return transformedDoc;
}
```

7. 创建 XML

一般创建 XML 是写文件前的工作，这就像 StringBuffer 一样容易。

```
  public Document createDocument() {
    Document document = DocumentHelper.createDocument();
    Element root = document.addElement(root);
    Element author1 =
        root
          .addElement(author)
          .addAttribute(name, James)
```

```
        .addAttribute(location, UK)
        .addText(James Strachan);
    Element author2 =
        root
        .addElement(author)
        .addAttribute(name, Bob)
        .addAttribute(location, US)
        .addText(Bob McWhirter);
    return document;
}
```

8. 文件输出

一个简单的输出方法是将一个Document 或任何的Node 通过write 方法输出

```
FileWriter out = new FileWriter( foo.xml );
document.write(out);
```

如果你想改变输出的格式，比如美化输出或缩减格式，可以用 XMLWriter 类

```
public void write(Document document) throws IOException {
    // 指定文件
    XMLWriter writer = new XMLWriter(
        new FileWriter( output.xml )
    );
    writer.write( document );
    writer.close();
    // 美化格式
    OutputFormat format = OutputFormat.createPrettyPrint();
    writer = new XMLWriter( System.out, format );
    writer.write( document );
    // 缩减格式
    format = OutputFormat.createCompactFormat();
    writer = new XMLWriter( System.out, format );
    writer.write( document );
}
```

如何，DOM4J 够简单吧，当然，还有一些复杂的应用没有提到，如 ElementHandler 等。如果你动心了，那就一起来用 DOM4J.

DOM4J 官方网站：(我老连不上)

http://www.dom4j.org/

DOM4J 下载(SourceForge)，最新版本为 1.4

http://sourceforge.net/projects/dom4j

用 Dom4j 解析 XML 及中文问题

发表于 2004 年 9 月 27 日 20:21

本文主要讨论了用 dom4j 解析 XML 的基础问题，包括建立 XML 文档，添加、修改、删除节点，以及格式化（美化）输出和中文问题。可作为 dom4j 的入门资料。

转载自:http://jalorsoft.com/holen/

作者：陈光（holen@263.net）

时间：2004-09-11

本文主要讨论了用 dom4j 解析 XML 的基础问题，包括建立 XML 文档，添加、修改、删除节点，以及格式化（美化）输出和中文问题。可作为 dom4j 的入门资料。

1. 下载与安装

dom4j 是 sourceforge.net 上的一个开源项目，主要用于对 XML 的解析。从 2001 年 7 月发布第一版以来，已陆续推出多个版本，目前最高版本为 1.5。

dom4j 专门针对 Java 开发，使用起来非常简单、直观，在 Java 界，dom4j 正迅速普及。

可以到 http://sourceforge.net/projects/dom4j 下载其最新版。

dom4j1.5 的完整版大约 13M，是一个名为 dom4j-1.5.zip 的压缩包，解压后有一个 dom4j-1.5.jar 文件，这就是应用时需要引入的类包，另外还有一个 jaxen-1.1-beta-4.jar 文件，一般也需要引入，否则执行时可能抛 java.lang.NoClassDefFoundError: org/jaxen/JaxenException 异常，其他的包可以选择用之。

2. 示例 XML 文档（holen.xml）

为了述说方便，先看一个 XML 文档，之后的操作均以此文档为基础。

holen.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <!--This is a test for dom4j, holen, 2004.9.11-->
    <book show="yes">
      <title>Dom4j Tutorials</title>
    </book>
    <book show="yes">
      <title>Lucene Studing</title>
    </book>
    <book show="no">
      <title>Lucene in Action</title>
    </book>
    <owner>O'Reilly</owner>
</books>
```

这是一个很简单的 XML 文档，场景是一个网上书店，有很多书，每本书有两个属性，一个是书名[title]，一个为是否展示[show]，最后还有一项是这些书的拥有者[owner]信息。

3. 建立一个 XML 文档

```
    /**
     * 建立一个 XML 文档,文档名由输入属性决定
     * @param filename 需建立的文件名
     * @return 返回操作结果, 0 表失败, 1 表成功
     */
    public int createXMLFile(String filename){
      /** 返回操作结果, 0 表失败, 1 表成功 */
      int returnValue = 0;
      /** 建立 document 对象 */
      Document document = DocumentHelper.createDocument();
      /** 建立 XML 文档的根 books */
      Element booksElement = document.addElement("books");
      /** 加入一行注释 */
      booksElement.addComment("This is a test for dom4j, holen,
2004.9.11");
      /** 加入第一个 book 节点 */
      Element bookElement = booksElement.addElement("book");
      /** 加入 show 属性内容 */
      bookElement.addAttribute("show","yes");
      /** 加入 title 节点 */
      Element titleElement = bookElement.addElement("title");
      /** 为 title 设置内容 */
      titleElement.setText("Dom4j Tutorials");

      /** 类似的完成后两个 book */
      bookElement = booksElement.addElement("book");
      bookElement.addAttribute("show","yes");
      titleElement = bookElement.addElement("title");
      titleElement.setText("Lucene Studing");
      bookElement = booksElement.addElement("book");
      bookElement.addAttribute("show","no");
      titleElement = bookElement.addElement("title");
      titleElement.setText("Lucene in Action");

      /** 加入 owner 节点 */
      Element ownerElement = booksElement.addElement("owner");
```

```
    ownerElement.setText("O'Reilly");

    try{
      /** 将 document 中的内容写入文件中 */
      XMLWriter writer = new XMLWriter(new FileWriter(new
File(filename)));
      writer.write(document);
      writer.close();
      /** 执行成功,需返回 1 */
      returnValue = 1;
    }catch(Exception ex){
      ex.printStackTrace();
    }

    return returnValue;
  }
```

说明：

Document document = DocumentHelper.createDocument();

通过这句定义一个 XML 文档对象。

Element booksElement = document.addElement("books");

通过这句定义一个 XML 元素，这里添加的是根节点。

Element 有几个重要的方法：

l addComment：添加注释

l addAttribute：添加属性

l addElement：添加子元素

最后通过 XMLWriter 生成物理文件，默认生成的 XML 文件排版格式比较乱，可以通过 OutputFormat 类的 createCompactFormat()方法或 createPrettyPrint()方法格式化输出，默认采用 createCompactFormat()方法，显示比较紧凑，这点将在后面详细谈到。

生成后的 holen.xml 文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<books><!--This is a test for dom4j, holen, 2004.9.11--><book
show="yes"><title>Dom4j Tutorials</title></book><book
show="yes"><title>Lucene Studing</title></book><book
show="no"><title>Lucene in
Action</title></book><owner>O'Reilly</owner></books>
```

4. 修改 XML 文档

有三项修改任务，依次为：

l 如果 book 节点中 show 属性的内容为 yes,则修改成 no

l 把 owner 项内容改为 Tshinghua，并添加 date 节点

l 若 title 内容为 Dom4j Tutorials,则删除该节点

```
    /**
     * 修改 XML 文件中内容,并另存为一个新文件
     * 重点掌握 dom4j 中如何添加节点,修改节点,删除节点
     * @param filename 修改对象文件
     * @param newfilename 修改后另存为该文件
     * @return 返回操作结果, 0 表失败, 1 表成功
     */
    public int ModiXMLFile(String filename,String newfilename){
      int returnValue = 0;
      try{
         SAXReader saxReader = new SAXReader();
         Document document = saxReader.read(new File(filename));
         /** 修改内容之一: 如果 book 节点中 show 属性的内容为 yes,则修改成 no
*/
         /** 先用 xpath 查找对象 */
         List list = document.selectNodes("/books/book/@show" );
         Iterator iter = list.iterator();
         while(iter.hasNext()){
           Attribute attribute = (Attribute)iter.next();
           if(attribute.getValue().equals("yes")){
              attribute.setValue("no");
           }
         }
```

```
        /**
         * 修改内容之二: 把 owner 项内容改为 Tshinghua
         * 并在 owner 节点中加入 date 节点,date 节点的内容为 2004-09-11,还为 date 节点添加一个属性 type
         */
        list = document.selectNodes("/books/owner" );
        iter = list.iterator();
        if(iter.hasNext()){
          Element ownerElement = (Element)iter.next();
          ownerElement.setText("Tshinghua");
          Element dateElement = ownerElement.addElement("date");
          dateElement.setText("2004-09-11");
          dateElement.addAttribute("type","Gregorian calendar");
        }

        /** 修改内容之三: 若 title 内容为 Dom4j Tutorials,则删除该节点 */
        list = document.selectNodes("/books/book");
        iter = list.iterator();
        while(iter.hasNext()){
          Element bookElement = (Element)iter.next();
          Iterator iterator = bookElement.elementIterator("title");
          while(iterator.hasNext()){
             Element titleElement=(Element)iterator.next();
             if(titleElement.getText().equals("Dom4j Tutorials")){
               bookElement.remove(titleElement);
             }
          }
        }

        try{
          /** 将 document 中的内容写入文件中 */
          XMLWriter writer = new XMLWriter(new FileWriter(new File(newfilename)));
          writer.write(document);
          writer.close();
          /** 执行成功,需返回 1 */
          returnValue = 1;
        }catch(Exception ex){
          ex.printStackTrace();
        }

      }catch(Exception ex){
        ex.printStackTrace();
      }
```

```
    return returnValue;
  }
```

说明：

```
List list = document.selectNodes("/books/book/@show" );

list = document.selectNodes("/books/book");
```

上述代码通过 xpath 查找到相应内容。

通过 setValue()、setText()修改节点内容。

通过 remove()删除节点或属性。

5. 格式化输出和指定编码

默认的输出方式为紧凑方式，默认编码为 UTF-8，但对于我们的应用而言，一般都要用到中文，并且希望显示时按自动缩进的方式的显示，这就需用到 OutputFormat 类。

```
/**
 * 格式化 XML 文档,并解决中文问题
 * @param filename
 * @return
 */
public int formatXMLFile(String filename){
  int returnValue = 0;
  try{
     SAXReader saxReader = new SAXReader();
     Document document = saxReader.read(new File(filename));
     XMLWriter writer = null;
     /** 格式化输出,类型 IE 浏览一样 */
     OutputFormat format = OutputFormat.createPrettyPrint();
     /** 指定 XML 编码 */
     format.setEncoding("GBK");
     writer= new XMLWriter(new FileWriter(new
File(filename)),format);
     writer.write(document);
     writer.close();
     /** 执行成功,需返回 1 */
     returnValue = 1;
  }catch(Exception ex){
     ex.printStackTrace();
  }
  return returnValue;
}
```

说明：

```
OutputFormat format = OutputFormat.createPrettyPrint();
```

这句指定了格式化的方式为缩进式，则非紧凑式。

```
format.setEncoding("GBK");
```

指定编码为 GBK。

```
XMLWriter writer = new XMLWriter(new FileWriter(new
File(filename)),format);
```

这与前面两个方法相比，多加了一个 OutputFormat 对象，用于指定显示和编码方式。

6. 完整的类代码

前面提出的方法都是零散的，下面给出完整类代码。

Dom4jDemo.java

```
package com.holen.dom4j;

import java.io.File;
import java.io.FileWriter;
import java.util.Iterator;
import java.util.List;

import org.dom4j.Attribute;
import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.SAXReader;
import org.dom4j.io.XMLWriter;

/**
 * @author Holen Chen
 */
public class Dom4jDemo {

    public Dom4jDemo() {
    }

    public int createXMLFile(String filename){…}
    public int ModiXMLFile(String filename,String newfilename){…}
    public int formatXMLFile(String filename){…}

    public static void main(String[] args) {
      Dom4jDemo temp = new Dom4jDemo();
      System.out.println(temp.createXMLFile("d://holen.xml"));        Syst
em.out.println(temp.ModiXMLFile("d://holen.xml","d://holen2.xml"));
      System.out.println(temp.formatXMLFile("d://holen2.xml"));
    }
}
```

说明：

main()方法中依次调用三个方法，第一个方法用于生成 holen.xml，第二个方法用于修改 holen.xml，并且修改后的内容另存为 holen2.xml，第三个方法将 holen2.xml 格式化缩进式输出，并指定编码方式为 GBK。

一个应用 Dom4j 的例子

[2005-4-21]

Created with Colorer-take5 Library. Type 'net.sf.colorer.FileType@777255'

```
 0: /*
 1:  * Created on 2005-4-19
 2:  *
 3:  * Copyright   (c) 2005 Julysea
 4:  * Window - Preferences - Java - Code Style - Code Templates
 5:  */
 6:
 7: /*应用此 log4j 的 log4j.properties 配置文件
 8:  *
 9:  *#################################################
################################
10:  *# Categories and levels
11:  *#################################################
################################
12:  *
13:  *log4j.rootCategory=DEBUG, FileApp,
ConApp
14:  *log4j.category.de.jayefem=DEBUG, FileApp,
ConApp
15:  *
16:  *#################################################
################################
17:  *# Appenders
18:  *#################################################
################################
19:  *
20:  *# ConApp is set to be a ConsoleAppender.
```

```
 21:  *log4j.appender.ConApp=org.apache.log4j.ConsoleAppender
 22:  *log4j.appender.ConApp.Target=System.out
 23:  *log4j.appender.ConApp.layout=org.apache.log4j.PatternLayout
 24:  *log4j.appender.ConApp.layout.ConversionPattern=%d{ABSOLUTE}
%5p %c{1}:%L - %m%n
 25:  *
 26:  *# FileApp
 27:  *log4j.appender.FileApp=org.apache.log4j.RollingFileAppender

 28:  *log4j.appender.FileApp.File=./log4e.log
 29:  *log4j.appender.FileApp.MaxFileSize=500KB

 30:  *# Keep one backup file
 31:  *log4j.appender.FileApp.MaxBackupIndex=1

 32:  *log4j.appender.FileApp.layout=org.apache.log4j.PatternLayout

 33:  *log4j.appender.FileApp.layout.ConversionPattern=%d [%t] %-5p
%c - %m%n
 34:  */
 35:
 36:
 37: /*应用此 XML 文件做测试
 38:  *
 39:  *<EW cmd="login" mod="Login" version="6.0">
 40:  *<Source uns="" type="user"/>
 41:  *<Username>zhangzhiyun@hp</Username>
 42:  *<Password>111111</Password>
 43:  *<Version>6.01.06.00</Version>
 44:  *</EW>
 45:  */
 46: package xml;
 47:
 48: import java.io.BufferedReader;
 49: import java.io.BufferedWriter;
 50: import java.io.File;
 51: import java.io.FileReader;
 52: import java.io.FileWriter;
 53: import java.io.IOException;
 54:
 55: import org.apache.log4j.Logger;
 56: import org.dom4j.Attribute;
 57: import org.dom4j.DocumentException;
 58: import org.dom4j.DocumentHelper;
```

```
59: import org.dom4j.Element;
60:
61: /**
62:  * @author julysea
63:  *
64:  * 一个用 Dom4j 解析 xml 的例子
65:  *
66:  */
67: public class Dom4jTest {
68:
69:     private static final Logger logger =
Logger.getLogger(Dom4jTest.class);
70:
71:     public static void main(String[] args) throws IOException,
72:             DocumentException {
73:         BufferedReader reader=new BufferedReader(new
FileReader("ew.xml"));
74:         String tempStr;
75:         String ewXml="";
76:         while((tempStr=reader.readLine())!=null) {
77:             ewXml=ewXml+tempStr;
78:             logger.debug(tempStr);
79:         }
80:         Element root = null;
81:
82:         root = DocumentHelper.parseText(ewXml).getRootElement();
83:         Attribute rootCmd=root.attribute("cmd");
84:         Attribute rootVersion=root.attribute("version");
85:         logger.debug("rootNmae = "+root.getName());
86:         logger.debug("EW'cmd = "+rootCmd.getValue());
87:         logger.debug("EW'version = "+rootVersion.getValue());
88:
89:         Element usrName=root.element("Username");
90:         logger.debug("EW.Username value =
"+usrName.getTextTrim());
91:
92:         Element source=root.element("Source");
93:         Attribute sourceUns=source.attribute("uns");
94:         logger.debug("EW.Source'uns"+sourceUns.getValue());
95:         Attribute sourceType=source.attribute("type");
96:         logger.debug("EW.Source'type = "+sourceType.getValue());
97:
98:
99:         //创建一个 Xml 文件
```

```
100:        Element user=DocumentHelper.createElement("User");
101:        user.addAttribute("type", "user");
102:        user.addElement("name").addAttribute("type",
"PinYin").setText("Julysea");
103:        user.addElement("age").setText("29");
104:        String oneXml=user.asXML();
105:
106:        BufferedWriter out=new BufferedWriter(new
FileWriter("oneXml.xml"));
107:        out.write(oneXml);
108:        out.close();
109:    }
110: }
```

Dom4j 编码问题彻底解决

这几天开始学习 dom4j，在网上找了篇文章就开干了，上手非常的快，但是发现了个问题就是无法以 UTF-8 保存 xml 文件，保存后再次读出的时候会报“Invalid byte 2 of 2-byte UTF-8 sequence.”这样一个错误，检查发现由 dom4j 生成的这个文件，在使用可正确处理 XML 编码的任何的编辑器中中文成乱码，从记事本查看并不会出现乱码会正确显示中文。让我很是头痛。试着使用 GBK、gb2312 编码来生成的 xml 文件却可以正常的被解析。因此怀疑的 dom4j 没有对 utf-8 编码进行处理。便开始查看 dom4j 的原代码。终于发现的问题所在，是自己程序的问题。

在 dom4j 的范例和网上流行的《DOM4J 使用简介》这篇教程中新建一个 xml 文档的代码都类似如下

```
public void createXML(String fileName) {
    document.nbspdoc = org.dom4j.document.elper.createdocument.);
```

```
        Element root = doc.addElement("book");
        root.addAttribute("name", "我的图书");

        Element childTmp;
        childTmp = root.addElement("price");
        childTmp.setText("21.22");

        Element writer = root.addElement("author");
        writer.setText("李四");
        writer.addAttribute("ID", "001");

        try {
            org.dom4j.io.XMLWriter xmlWriter = new org.dom4j.io.XMLWr
iter(
                    new FileWriter(fileName));
            xmlWriter.write(doc);
            xmlWriter.close();
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
```

在上面的代码中输出使用的是 FileWriter 对象进行文件的输出。这就是不能正确进行文件编码的原因所在，java 中由 Writer 类继承下来的子类没有提供编码格式处理，所以 dom4j 也就无法对输出的文件进行正确的格式处理。这时候所保存的文件会以系统的默认编码对文件进行保存，在中文版的 window 下 java 的默认的编码为 GBK，也就是所虽然我们标识了要将 xml 保存为 utf-8 格式但实际上文件是以 GBK 格式来保存的,所以这也就是为什么能够我们使用 GBK、GB2312 编码来生成 xml 文件能正确的被解析，而以 UTF-8 格式生成的文件不能被 xml

解析器所解析的原因。

好了现在我们找到了原因所在了，我们来找解决办法吧。首先我们看看 dom4j 是如何实现编码处理的

```
   public XMLWriter(OutputStream out) throws UnsupportedEncodingExcep
tion {
        //System.out.println("In OutputStream");
        this.format = DEFAULT_FORMAT;
        this.writer = createWriter(out, format.getEncoding());
        this.autoFlush = true;
       namespaceStack.push(Namespace.NO_NAMESPACE);
    }

    public XMLWriter(OutputStream out, OutputFormat format) throws Un
supportedEncodingException {
        //System.out.println("In OutputStream,OutputFormat");
        this.format = format;
        this.writer = createWriter(out, format.getEncoding());
        this.autoFlush = true;
       namespaceStack.push(Namespace.NO_NAMESPACE);
    }
    /**
     * Get an OutputStreamWriter, use preferred encoding.
     */
    protected Writer createWriter(OutputStream outStream, String enco
ding) throws UnsupportedEncodingException {
        return new BufferedWriter(
            new OutputStreamWriter( outStream, encoding )
        );
    }
```

由上面的代码我们可以看出 dom4j 对编码并没有进行什么很复杂的处理，完全通过 java 本身的功能来完成。所以我们在使用 dom4j 的来生成我们的 XML 文件时不应该直接为在构建 XMLWriter 时，不应该直接为其赋一个 Writer 对象，而应该通过一个 OutputStream 的子类对象来构建。也就是说在我们上面的代码中，不应该用 FileWriter 对象来构建 xml 文档，而应该使用 FileOutputStream 对象来构建所以将代码修改入下：

```
    public void createXML(String fileName) {
        document.nbspdoc = org.dom4j.document.elper.createdocument.);
        Element root = doc.addElement("book");
        root.addAttribute("name", "我的图书");

        Element childTmp;
        childTmp = root.addElement("price");
        childTmp.setText("21.22");

        Element writer = root.addElement("author");
        writer.setText("李四");
        writer.addAttribute("ID", "001");

        try {
            //注意这里的修改
            org.dom4j.io.XMLWriter xmlWriter = new org.dom4j.io.XMLWr
iter(
                    new FileOutputStream(fileName));
            xmlWriter.write(doc);
            xmlWriter.close();
        }
        catch (Exception e) {
            System.out.println(e);
```

```
        }
    }
```

至此 DOM4J 的问题编码问题算是告一段落，希望对此文章对其他朋友有用。

Dom4j 的基本使用

下载 dom4j 后，在其文档中就用详细的使用说明，我又将其封装了一下：

```
package org.tju.msnrl.butil;

import java.io.*;
import java.util.*;
import org.dom4j.*;
import org.dom4j.io.XMLWriter;
import org.dom4j.io.SAXReader;

/**
 * Dom4j 封装类
 * <p>Title: 天津大学博士后流动站</p>
 * <p>Description: 天津大学人事处制作维护</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: 天津大学软件学院.NET 实验室（MSNRL）</p>
 * @author Jonathan Q. Bo
 * @version 1.0
 */

public class BDom4j {
  /**XML 文件路径*/
```

```
private String XMLPath = null;
/**XML 文档*/
private Document document = null;

public BDom4j() {
}

/**
 * 初始化 xml 文件
 * @param XMLPath 文件路径
 */
public BDom4j(String XMLPath){
  this.XMLPath = XMLPath;
}

/**
 * 打开文档
 */
public void openXML(){
  try{
    SAXReader reader = new SAXReader();
    this.document = reader.read(this.XMLPath);
    System.out.println("openXML() successful ...");
  }catch(Exception e){
    System.out.println("openXML() Exception:" + e.getMessage());
  }
}

/**
```

```
 * 创建文档
 * @param rootName 根节点名称
 */
public void createXML(String rootName){
  try{
    this.document = DocumentHelper.createDocument();
    Element root = document.addElement(rootName);
    System.out.println("createXML() successful...");
  }catch(Exception e){
    System.out.println("createXML() Exception:" + e.getMessage());
  }
}

/**
 * 添加根节点的child
 * @param nodeName 节点名
 * @param nodeValue 节点值
 */
public void addNodeFromRoot(String nodeName, String nodeValue){
  Element root = this.document.getRootElement();
  Element level1 = root.addElement(nodeName);
  level1.addText(nodeValue);
}

/**
 * 打开文档
 * @param filePath 文档路径
 */
public void openXML(String filePath){
```

```
        try{
          SAXReader saxReader = new SAXReader();
          this.document = saxReader.read(filePath);
          System.out.println("openXML(String filePath) successful ...");
        }catch(Exception e){
          System.out.println("openXML() Exception:" + e.getMessage());
        }
      }

      /**
       * 保存文档
       */
      public void saveXML(){
        try{
          XMLWriter output = new XMLWriter(new FileWriter(new
    File(this.XMLPath)));
          output.write(document);
          output.close();
          System.out.println("saveXML() successful ...");
        }catch(Exception e1){
          System.out.println("saveXML() Exception:" + e1.getMessage());
        }
      }

      /**
       * 保存文档
       * @param toFilePath 保存路径
       */
      public void saveXML(String toFilePath) {
```

```
    try {
      XMLWriter output = new XMLWriter(new FileWriter(new
File(toFilePath)));
      output.write(document);
      output.close();
    }
    catch (Exception e1) {
      System.out.println("saveXML() Exception:" + e1.getMessage());
    }
  }


  /**
   * 获得某个节点的值
   * @param nodeName 节点名称
   */
  public String getElementValue(String nodeName){
    try {
      Node node = document.selectSingleNode("//" + nodeName);
      return node.getText();
    }
    catch (Exception e1) {
      System.out.println("getElementValue() Exception: " +
e1.getMessage());
      return null;
    }
  }
```

```
  /**
   * 获得某个节点的子节点的值
   * @param nodeName
   * @param childNodeName
   * @return
   */
  public String getElementValue(String nodeName, String childNodeName){
    try {
      Node node = this.document.selectSingleNode("//" + nodeName + "/"
+ childNodeName);
      return node.getText();
    }
    catch (Exception e1) {
      System.out.println("getElementValue() Exception: " +
e1.getMessage());
      return null;
    }
  }

  /**
   * 设置一个节点的 text
   * @param nodeName 节点名
   * @param nodeValue 节点值
   */
  public void setElementValue(String nodeName, String nodeValue){
    try{
      Node node = this.document.selectSingleNode("//" + nodeName);
      node.setText(nodeValue);
    }catch(Exception e1){
```

```
      System.out.println("setElementValue() Exception:" +
e1.getMessage());
    }
  }

  /**
   * 设置一个节点值
   * @param nodeName 父节点名
   * @param childNodeName 节点名
   * @param nodeValue 节点值
   */
  public void setElementValue(String nodeName, String childNodeName,
                              String nodeValue) {
    try {
      Node node = this.document.selectSingleNode("//" + nodeName + "/"
+ childNodeName);
      node.setText(nodeValue);
    }
    catch (Exception e1) {
      System.out.println("setElementValue() Exception:" +
e1.getMessage());
    }
  }

}
```

简单封装后，可以用来读写 XML 文档，对网站进行配置：如指定页面整体风格的 css 文件，在 context init 时读取存入 context 中，在页面中通过读取 context 中的相应属性来确定 css 文件名，完成一项配置，其它的动态配置都类似；

```
public class BListener extends HttpServlet implements
ServletContextListener, ServletContextAttributeListener,
HttpSessionListener, HttpSessionAttributeListener {
  private static String XML_FILE_PATH = "c:/test.xml";

  //Notification that the web application is ready to process requests
  public void contextInitialized(ServletContextEvent sce) {
    BDom4j xmlmng = new BDom4j(XML_FILE_PATH);
    xmlmng.openXML();
    sce.getServletContext().setAttribute("css",xmlmng.getElementValue
("style-sheet"));
    System.out.println("### context initialized...");
  }
```