# JAVA und XSLT

*XML-Dokumente mit JDOM transformieren*

**Alfred Nussbaumer**

Wie XML-Dokumente mittels DOM und SAX verarbeitet werden können, wurde in einigen Beispielen in PCNEWS-87 ([1]) und PCNEWS-89 ([2]) gezeigt. In diesem Beitrag soll JDOM vorgestellt werden: Mit JDOM stehen JAVA-Klasssen zur Verfügung, mit denen XML-Daten komfortabel ausgegeben und bearbeitet werden können. Konkret sollen XML-Dokumente transformiert werden, und zwar im Sinn von XSL (*XML Stylesheet Language*). XSL besteht aus drei Bereichen: XSLT (*XSL Transformations*), XPATH und XSL-FO (*XSL Formatting Objects*). Wie Sie JDOM beziehen und einfache Transformationen durchführen können ist Inhalt dieses Artikels.

## 1. JDOM vorbereiten

JDOM wird von der Adresse `http://www.jdom.org` als komprimiertes Dateiarchiv bezogen; für die folgenden Beispiele wurde `jdom-b9.tar.gz` verwendet. Nach dem Entpacken können die Skript-Dateien `build.bat` (für Windows) und `build.sh` (für Linux) ausgeführt werden. Mit ihnen werden zusätzliche Dateien erstellt. Im einfachsten Fall kopieren Sie die Datei `jdom.jar` in das Verzeichnis `$JAVA_HOME/jre/lib/ext`, oder Sie geben den Pfad zu `jdom.jar` als `CLASS_PATH` an.

Jedenfalls sollten Sie die API-Dokumente durchsehen, die im Verzeichnis `build/apidocs` zu finden sind. In ihnen werden alle Klassen (analog zur Java-Dokumentation ([4]) behandelt.

Für das folgende Beispiel (und für die weiteren) verwenden wir die einfache XML-Datei „`weblinks.xml`":

**weblinks.xml**

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE weblinks SYSTEM "weblinks.dtd">
<weblinks>
  <eintrag id="0">
    <kategorie>edv</kategorie>
    <url>http://www.w3.org</url>
    <notiz>W3-Konsortium</notiz>
    <notiz>Technische Referenz</notiz>
  </eintrag>
  <eintrag id="1">
    <kategorie>phy</kategorie>
    <url>http://www.cern.ch</url>
    <notiz>Europaeisches Kernforschungszentrum</notiz>
    <notiz>Aktuelles zur Hochenergiephysik</notiz>
    <notiz>Materialien zur Elementarteilchenphysik</notiz>
  </eintrag>
  ...
</weblinks>
```

Die XML-Datei wird gegen folgende DTD (*Document Type Definition*, vgl. [5]) „`weblinks.dtd`" validiert:

```
<!ELEMENT weblinks (eintrag*)>
 <!ELEMENT eintrag (kategorie, url, notiz*, datum?)>
 <!ATTLIST eintrag id CDATA #REQUIRED>
  <!ELEMENT kategorie (#PCDATA)>
  <!ELEMENT url (#PCDATA)>
  <!ELEMENT notiz (#PCDATA)>
  <!ELEMENT datum (#PCDATA)>
```

## 2. XML-Dokumente ausgeben

Im ersten Beispiel (`jdom1.java`) geben wir lediglich den kompletten Inhalt einer XML-Datei aus:

**jdom1.java**

```
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;

public class jdom1 {
  public static void main (String args[]) {
    try {
      SAXBuilder builder = new SAXBuilder();
      Document document = builder.build("weblinks.xml");
      XMLOutputter ausgabe = new XMLOutputter();
      ausgabe.output(document, System.out);
    }
    catch (JDOMException je) {
      System.out.println("JDOM-Fehler: " + je.getMessage());
    }
    catch (Exception ie) {
      System.out.println(ie.getMessage());
    }
  }
}
```

Das angegebene XML-Dokument wird mit der `SAXBuilder`-Methode `build()` im Arbeitsspeicher abgebildet. Dazu wird im Beispiel das Document-Objekt `document` verwendet. Das Package `input` stellt die erforderlichen Klassen zur Verfügung. Die Klassen für die Ausgabe sind im Package `output` enthalten - die Klasse `XMLOutputter` erlaubt die einfache Ausgabe der XML-Daten; ihre Methode `output()` gibt das angegebene Document-Objekt an den festgelegten Datenstrom aus. Im Beispiel wird das Dokument über die Variable `document` referenziert; die Ausgabe erfolgt auf der Konsole. Wir erhalten:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblinks>
   <eintrag id="0">
        <kategorie>edv</kategorie>
        <url>http://www.w3.org</url>
        <notiz>W3-Konsortium</notiz>
        <notiz>Technische Referenz</notiz>
   </eintrag>
   <eintrag id="1">
        <kategorie>phy</kategorie>
        <url>http://www.cern.ch</url>
...
```

Details zu allen JDOM-Klassen sind in der Dokumentation enthalten (vgl. [5]).

Mit JDOM lassen sich bestimmte Elemente oder Attribute einer XML-Datei auswählen oder neue Elemente können eingefügt werden (vgl. dazu **„6 Aufgaben, Ausblick"**).

## 3. XSLT - Grundlagen

Mit XSLT kann eine XML-Datei nach bestimmten Regeln in eine andere XML-Datei (oder eine beliebige Textdatei, z.B. LaTeX, vgl. [8], [9]) umgeformt werden. Zur Transformation ist ein so genannter XSLT-Prozessor nötig: Aktuelle Webbrowser enthalten einen solchen Prozessor. Auf der Kommandozeile können Sie beispielsweise XT (siehe [6]), XALAN (vgl. [7]) oder SAXON einsetzen. SAXON ist Teil aktueller Linux-Distributionen; das folgende Beispiel wurde mit Hilfe von SAXON durchgeführt.

Alle Transformationsregeln werden in einer eigenen XML-Datei (Stylesheet-Datei) zusammengefasst, die aus XSL-Elementen aufgebaut wird. Das folgende Beispiel zeigt, wie der Inhalt einer XML-Datei in eine HTML-Datei umgeformt wird:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl =
"http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <head>
        <title>WebLinks</title>
      </head>
      <body>
        <h1>WebLinks</h1>
        <xsl:apply-templates select="weblinks/eintrag">
          <xsl:sort select="kategorie" />
        </xsl:apply-templates>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="weblinks/eintrag">
    <h2><xsl:value-of select="kategorie" /></h2>
    <p>
      <xsl:apply-templates select="url" />
    </p>
  </xsl:template>

  <xsl:template match="url">
    <strong><xsl:value-of select="../url" /></strong>
    <xsl:apply-templates select="../notiz" />
```

alfred.nussbaumer@schule.at

```
  </xsl:template>

  <xsl:template match="notiz">
    <dd>
      <xsl:value-of select="." />
    </dd>
  </xsl:template>
</xsl:stylesheet>
```

In dieser (wohlgeformten) XML-Datei kommen nach der XML-Deklaration HTML-Elemente und XSL-Elemente vor. Um eine Verwechslung auszuschließen, werden so genannte Namensräume verwendet: Der Namensraum für die XSL-Elemente wird durch das Präfix `xsl` angegeben.

Wie XSL-Transformationen ablaufen, ist gewöhnungsbedürftig: Grundsätzlich bestimmten *Templates* (Schablonen, Vorlagen), welche Elemente für die Ausgabe gewählt werden und welche Elemente neu generiert werden. Mit Hilfe der angegebenen Muster (*patterns*) wird festgelegt, auf welche Elemente des Eingabedokumentes ein Template angewendet werden soll. Was mit diesem Element geschehen soll wird in den so genannten Vorlagenregeln (*template rules*) festgelegt, die zwischen dem Starttag und Endetag des `<xsl:template>`-Elements stehen.

Das erste Template im Beispiel wird anhand des Musters „/“ auf das Wurzelelement der XML-Datei `„weblinks.xml"` angewendet. Konkret werden einige HTML-Elemente ausgegeben und ein Template ausgeführt (`<xsl:apply-templates>`), das alle `<eintrag>`-Elemente anhand des Inhaltes des `<kategorie>`-Elementes alphabetisch sortiert (`<xsl:sort>`). Weiters legt das `select`-Attribut des `<xsl:apply-templates>` fest, welche weiteren Templates ausgeführt werden sollen: Hier soll das Template, das auf das Muster `„weblinks/eintrag"` passt ausgeführt werden.

Das zweite Template bezieht sich genau auf dieses Muster: Hier soll zunächst das HTML-Element `<h2>` ausgegeben werden, dessen Inhalt aus dem Textwert des `<kategorie>`-Elementes besteht. An dieser Stelle ist es wichtig, dass das `<kategorie>`-Element ein direktes Kindelement von `<eintrag>` ist, welches auf Grund des Musters `„weblinks/eintrag"` ausgewählt wurde. Der Textwert wird mit Hilfe des XSL-Elementes `<xsl:value-of>` ausgegeben. Im folgenden HTML-Absatz (`<p>`) soll das Template für das Muster `„url"` ausgewählt werden.

Im vorletzten Template für das Muster `„url"` wird der Textwert des `<url>`-Elementes innerhalb des HTML-Elements `<strong>` ausgegeben. Anschließend wird das letzte Template aufgerufen, das auf das Muster `„notiz"` passt. Da der Pfad für das Muster korrekt angegeben werden muss, ist es innerhalb des `„url"`-Templates notwendig, mit dem Muster `„.."` eine Elementebene höher zu steigen.

Das letzte Template passt auf die `<notiz>`-Elemente: Der Textwert jedes `<notiz>`-Elementes wird zwischen `<dd>`-Tags ausgegeben. Alle Notizelemente werden mit Hilfe des Musters `„."` gewählt.

Zu Testzwecken führen wir die Transformation auf der Konsole aus; dabei wird die Ausgabe in die Datei `„weblinks.html"` umgeleitet:

```
$ saxon weblinks.xml weblinks.xsl > weblinks.html
```

Betrachtet man die erhaltene HTML-Datei mit einem Browser, so erhält man eine Ausgabe, die durch die verwendeten HTML-Elemente vorgegeben wird (siehe **Abbildung**).

Mit entsprechenden Stylesheet-Angaben in einer CSS-Datei erhält man im Browser eine entsprechend formatierte Ausgabe.

## 4. Transformation mit JDOM

JDOM verwendet zum Transformieren von XML-Dateien den XSLT-Prozessor XALAN. Dieser wird durch die Java API for XML Parsing (JAXP) zur Verfügung gestellt.

Das JDOM-Package transform enthält die Klassen zur Transformation: Die Klasse `JDOMSource` enthält die Methoden zum Einlesen eines XML-Dokumentes und bereitet das Dokument für die XSL-Transformtion vor. `JDOMResult` nimmt das Transformationsergebnis in Form einer Liste entgegen.

Das Beispiel `jdom2.java` zeigt eine einfache Anwendung der JDOM-Transformationsklassen:

**jdom2.java**

```
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.transform.JDOMSource;
import org.jdom.transform.JDOMResult;

import java.util.List;
```

# WebLinks

## ast

### http://www.nasa.gov

Amerikanische Weltraumfahrtbehoerde
Aktuelle Raumfahrtprojekte
Historische Daten

## ast

### http://www.heavens-above.com

Der abendliche Sternhimmel ...
Beobachtung von Satelliten
Iridium Flares
Mond- und Sonnenbahn

## edv

### http://www.w3.org

W3-Konsortium
Technische Referenz

## edv

*Ausgabe der Datei* `weblinks.html` *in einem Browser*

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

public class jdom2 {

  static List ergebnis;

  public static void main (String args[]) throws Exception {
    SAXBuilder builder = new SAXBuilder();
    Document document = builder.build("weblinks.xml");
    ergebnis = transform(document, "weblinks.xsl");
    XMLOutputter xmlausgabe = new XMLOutputter();
    xmlausgabe.output(ergebnis, System.out);
  }

  public static List transform(Document in, String stylesheet)
    throws JDOMException {
    try {
      Transformer transformer =
        TransformerFactory.newInstance().newTransformer
          (new StreamSource(stylesheet));
      JDOMResult ausgabe = new JDOMResult();
      transformer.transform(new JDOMSource(in), ausgabe);
      return ausgabe.getResult();
    }
    catch (TransformerException e) {
      throw new JDOMException(e);
    }
  }
}
```

Die Transformation geschieht innerhalb der Methode `transform()`: Das ursprüngliche XML-Dokument wird als Document-Variable übergeben; die Stylesheet-Datei mit den Transformationsregeln wird über ihren Dateinamen referenziert. `transform()` gibt das Ergebnis der Transformation als Liste zurück. Sie wird im Beispiel mit Hilfe der `XMLOutputter`-Methode `output()` auf der Konsole ausgegeben.

## 5. Ausgabedokument in einer Textdatei speichern

Der Ergebnisbaum wird im Beispiel `jdom2.java` als Abfolge von XML-Elementen (also „serialisiert“) auf der Konsole ausgegeben. Für komplexere Aufgaben ist es notwendig, die Ausgabe in einem neuen XML-Dokument zu speichern. Dazu importieren wir zunächst das Package `java.io` und erzeugen das File-Objekt `ausgabedatei`.

**jdom3.java**

```
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.transform.JDOMSource;
import org.jdom.transform.JDOMResult;

import java.util.List;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;

public class jdom3 {
  static List ergebnis;
  static File ausgabedatei = new File("weblinks_out.html");

  public static void main (String args[]) throws Exception {
    SAXBuilder builder = new SAXBuilder();
    Document document = builder.build("weblinks.xml");
    ergebnis = transform(document, "weblinks.xsl");
    XMLOutputter xmlausgabe = new XMLOutputter();
    try {
      FileOutputStream ausgabestrom =
        new FileOutputStream(ausgabedatei);
      DataOutputStream datei = new DataOutputStream(ausgabestrom);
      xmlausgabe.output(ergebnis, datei);
    }
    catch (IOException e) {
      System.out.println(e);
    }
  }

  public static List transform(Document in, String stylesheet)
    throws JDOMException {
    ...
  }
}
```

Die Methode `transform()` wird so wie in Beispiel `jdom2.java` verwendet.

## 6. Aufgaben, Ausblick

1. Bestimmte Elemente einer XML-Datei sind mit Hilfe der JDOM-Methoden auszuwählen. Falls vorhanden sollen die Attribute und Attributwerte und die Elementinhalte ausgegeben werden.

2. Ein neues XML-Dokument ist mit den JDOM-Methoden `addContent(Element)` und `setAttribute(Attributname, Attributwert)` zu erzeugen (Hinweis:

```
Element wurzelelement = new Element("wurzel");
wurzelelement.setAttribute("id","1");

Element kindelement = new Element("kind");
kindelement.addContent("Textinhalt");

wurzelelement.addContent(kindelement);

Document dokument = new Document(wurzelelement);
```

liefert den einfachen XML-Datenbaum

```
<?xml version="1.0" encoding="UTF-8"?>
<wurzel id="1"><kind>Textinhalt</kind></wurzel>
```

(vgl. dazu auch [13]).

## 7. Literatur, Weblinks


[1] „JAVA und DOM“, PCNEWS-87 April 2004, S. 26

[2] „JAVA und SAX“, PCNEWS-89 September 2004,

[3] http://www.s3.org/TR/REC-xml (W3C-Empfehlung zu XML, Version 1.0)

[4] http://java.sun.com/j2se/1.4.2/docs/index.html (Dokumentation aller verfügbaren Packages)

[5] JDOM-Dokumentation (im Verzeichnis `build/apidocs` der entpackten JDOM-Distribution enthalten)

[6] http://www.blnz.com/xt/index.html (Homepage für den XSLT-Prozessor XT)

[7] http://xml.apache.org/xalan-j/ (XSLT-Prozessor XALAN im Rahmen des Apache-Projekts)

[8] August Mistlbacher, Alfred Nussbaumer, „XML Ge-Packt“, mitp-Verlag

[9] August Mistlbacher, Alfred Nussbaumer, „XML Ent-Packt“, mitp-Verlag

[10] Herbert Schildt, „Java 2 Ent-Packt“, mitp-Verlag

[11] Christian Ullenboom, „Java ist auch eine Insel“, Galileo Computing

[12] http://www.gymmelk.ac.at/nus/informatik/xmlneu/ (Unterrichtsbeispiele zu XML)

[13] http://nus.lugsp.at/informatik/wpf/JAVA (Unterrichtsbeispiele zum Programmieren mit JAVA)