JAVA und DOM

XML-Dokumente verarbeiten

Alfred Nussbaumer

Seit Ende der 90er Jahre wurde XML (*Extensible Markup Language*) vom W3-Konsortium als Standard für eine flexible Auszeichnungssprache definiert. Zahlreiche Anwendungen speichern Informationen mittlerweile in Form von so genannten XML-Dokumenten. Wie JAVA XML-Daten auslesen und verarbeiten kann soll in einigen Beiträgen behandelt werden. In diesem ersten Artikel werden das DOM (*Document Object Model*) und grundlegende JAVA-Funktionen in einigen Beispielen vorgestellt. Die verwendeten Klassen sind seit dem JDK 1.4 Bestandteile von JAVA.

1. XML

Um XML zu verstehen, wird meistens die Verwandtschaft zu HTML zitiert: Ähnlich wie alle HTML-Objekte mit Hilfe geeigneter Auszeichnungselemente („*Tags*“) bezeichnet werden, werden alle XML-Elemente mit Anfangs- und Ende-Tags angegeben. Während HTML (und sein XML-Pendant XHTML) für die Verwendung von Browsern, PDAs und Mobiltelefonen entwickelt wurde, können XML-Dokumente universell eingesetzt werden. So speichern StarOffice ab der Version 6.0 und MS-Office ab der Version 2003 alle Dokumentdaten im XML-Format. Das W3-Konsortium hat eine genaue Spezifikation zu XML verabschiedet (***[1]***), zahlreiche Bücher enthalten detaillierte Informationen zu XML (z.B. ***[4]***, ***[5]***).

Für die folgenden JAVA-Beispiele verwenden wir folgende XML-Datei `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>
```

Wir erkennen eine wohlgeformte XML-Datei, bei der das Wurzelelement `<weblinks>` alle `<eintrag>`-Elemente und deren Kindelemente korrekt geschachtelt enthält. Die XML-Datei wird gegen folgende DTD (*Document Type Definition*) `weblinks.dtd` validiert:

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

In der DTD wird festgelegt, welche Elemente in den Dokumentenbaum eingefügt werden können. Für die obige, sehr einfache DTD gilt: Das Wurzelelement `weblinks` darf beliebig viele eintrag-Elemente enthalten; jedes `eintrag`-Element enthält das obligate Attribut `id`, ein `kategorie`-, ein `url`- und beliebig viele `notiz`-Elemente.

Um bestimmte Elemente (oder Attribute) einer XML-Datei auszuwählen, muss man den so genannten Dokumentenbaum vom Wurzelelement ausgehend durchsuchen. Eine bestimmte Abfrage liefert die Knoten (*nodes*), anhand derer die gewünschten Elemente genau bestimmt werden.

2. Grundlagen

Um eine XML-Datei parsen zu können benötigen Sie die Klassen `DocumentBuilderFactory` und `DocumentBuilder`, sowie die Interfaces `Document`, `Node` und `NodeList`. Sie sind in den Packages `javax.xml.parsers` und `org.w3c.dom` enthalten; ihre genaue Beschreibung ist in der JAVA-Dokumentation (***[3]***) angegeben.

Um mit dem DOM-Parser eine XML-Datei zu parsen sind schließlich vier Schritte nötig:

1. Eine neue Instanz der Klasse `DocumentBuilderFactory` erzeugen.

```
DocumentBuilderFactory factory =
     DocumentBuilderFactory.newInstance();
```

2. Eine Instanz der Klasse `DocumentBuilder` erzeugen.

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

3. Den XML-Dokumentenbaum parsen und ein `document`-Objekt erzeugen.

```
Document document = builder.parse("weblinks.xml");
```

4. Die gewünschten Elemente mit geeigneten DOM-Befehlen auswählen.

Das erste Beispiel `dom1.java` gibt alle Kindelemente des Wurzelelementes der Datei `weblinks.xml` aus:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Node;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

public class dom1 {
  public static void main (String args[]) throws Exception {
    DocumentBuilderFactory factory =
         DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse("weblinks.xml");
    NodeList KnotenListe = document.getElementsByTagName("url");
    int anzahl = KnotenListe.getLength();
    for (int i = 0; i < anzahl; i++) {
      System.out.println
        (KnotenListe.item(i).getFirstChild().getNodeValue());
    }
  }
}
```

Die gewünschte Knotenliste wird mit Hilfe der Methode `getElementsByTagName()` erhalten. Aus ihr werden innerhalb der Zählschleife nacheinander alle Knoten ausgewählt. Für jeden Knoten wählt man mit der Methode `getFirstChild()` den ersten Kindknoten (das ist in `weblinks.xml` jeweils der zum Element `url` enthaltene Text). Den Zeichenkettenwert dieses Textknotens erhält man schließlich mit der Methode `getNodeValue()`. Das Ergebnis ist eine einfache Liste der gespeicherten Webadressen:

```
http://www.w3.org
http://www.cern.ch
...
```

Ist der Knoten wie im vorliegenden Fall ein Textknoten, so kann die Methode `getNodeValue()` auch wegbleiben. Aufschlussreich ist jedenfalls die folgende Ausgabe:

```
    for (int i = 0; i < anzahl; i++) {
      System.out.println(KnotenListe.item(i));
    }
```

Damit erhält man:

```
<url>http://www.w3.org</url>
<url>http://www.cern.ch</url>
...
```

3. Elemente und Attribute anzeigen

Im Beispiel `dom1.java` wurden aus dem gesamten Dokumentenbaum alle Elemente mit der Bezeichnung `url` ausgewählt. Nun

alfred.nussbaumer@schule.at

sollen für alle Elemente `eintrag` einige Kindelemente ausgegeben werden: Wir geben die Geschwisterelemente `kategorie`, `url` und `notiz` aus:

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class dom2 {
  public static void main (String args[]) throws Exception {
    Node Knoten;
    DocumentBuilderFactory factory =
          DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse("weblinks.xml");
    NodeList KnotenListe =
          document.getElementsByTagName("eintrag");

    int anzahl = KnotenListe.getLength();
    for (int i = 0; i < anzahl; i++) {
      Knoten = KnotenListe.item(i);
      System.out.print(Knoten.getChildNodes().
          item(1).getFirstChild() + "\t");
      System.out.print(Knoten.getChildNodes().
          item(3).getFirstChild() + "\t");
      System.out.println(Knoten.getChildNodes().
          item(5).getFirstChild());
    }
  }
}
```

Die Auswahl der korrekten Knoten ist in diesem Beispiel etwas verzwickter: Zunächst werden alle Knoten mit dem Elementnamen `eintrag` ausgewählt. Zu jedem Knoten aus dieser Liste werden nun alle Kindknoten bestimmt, und aus diesen der 1., 3. und 5. Eintrag. Was sind nun die dazwischen liegenden Knoten? Die Lösung sieht man in der dem Beispiel zugrunde liegenden XML-Datei: Der besseren Lesbarkeit halber wurden die einzelnen Elemente mit Zeilenschaltungen und Einrückungen (Tabulatoren) gespeichert. Dieser so genannte Leerraum (*white space*) bildet nun jeweils einen Geschwisterknoten mit nicht relevantem Inhalt – die Methode `getFirstChild()` würde für den *white space* am 0., 2. und 4. Knoten den Wert `null` zurückgeben.

Mit der korrekten Knotenauswahl erhalten wir:

```
edv   http://www.w3.org    W3-Konsortium
phy   http://www.cern.ch   Europaeisches Kernforschungszentrum
...
```

Durch entsprechende Schachtelung lassen sich bestimmte Attribute und Elemente ausgeben. Im nächsten Beispiel werden Elemente und das `id`-Attribut ausgegeben und mit Hilfe von Tabulatoren, Zeilenschaltungen und gleichbleibenden Zeichenketten einfach formatiert:

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class dom3 {
  public static void main (String args[]) throws Exception {
    DocumentBuilderFactory factory =
              DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document xmlbaum = builder.parse("weblinks.xml");
    NodeList eintragKnoten = xmlbaum.getElementsByTagName("eintrag");

    int anzahl = eintragKnoten.getLength();
    for (int i = 0; i < anzahl; i++) {
      Element eintrag = (Element) eintragKnoten.item(i);
      String attribut = eintrag.getAttribute("id");
      System.out.print(attribut + ": ");

      NodeList urlKnoten = eintrag.getElementsByTagName("url");
      System.out.println
              (urlKnoten.item(0).getFirstChild().getNodeValue());
      System.out.println("--------------------");

      NodeList notizKnoten = eintrag.getElementsByTagName("notiz");
      int notizanzahl = notizKnoten.getLength();
      for (int j = 0; j < notizanzahl; j++) {
        System.out.print("\t o ");
        System.out.println
              (notizKnoten.item(j).getFirstChild().getNodeValue());
      }
      System.out.println("====================================\n");
    }
  }
}
```

Im Ergebnis lesen wir nach dem `id`-Attributwert den Zeichenkettenwert des `url`-Elements und anschließend alle Zeichenkettenwerte der `notiz`-Elemente:

```
0: http://www.w3.org
--------------------
      o W3-Konsortium
      o Technische Referenz
====================================
1: http://www.cern.ch
   ------------------
      o Europaeisches Kernforschungszentrum
      o Aktuelles zur Hochenergiephysik
      o Materialien zur Elementarteilchenphysik
====================================
2: http://www.nasa.gov
   --------------------
...
```

4. Eine neue XML-Datei erzeugen

Im letzten Beispiel soll aus der vorgegebenen XML-Datei `weblinks.xml` eine neue XML-Datei gebildet werden, die nur die Einträge einer bestimmten Kategorie enthält. Dazu ist es zunächst nötig den neuen XML-Baum `ausgabebaum` aufzubauen. Abschließend muss der gesamte Inhalt des neuen Dokumentenbaumes in einer Textdatei gespeichert werden.

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.io.*;

public class dom4 {
  public static void main (String args[]) throws Exception {
    DocumentBuilderFactory factory =
              DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document xmlbaum = builder.parse("weblinks.xml");
    Document ausgabebaum = builder.newDocument();
    Element wurzel = ausgabebaum.createElement("adressen");
    Element eintrag;
    Element neuElement;

    ausgabebaum.appendChild(wurzel);

    NodeList eintragKnoten = xmlbaum.getElementsByTagName("eintrag");
    int anzahl = eintragKnoten.getLength();
    for (int i = 0; i < anzahl; i++) {
      eintrag = (Element) eintragKnoten.item(i);
      NodeList kategorieKnoten =
            eintrag.getElementsByTagName("kategorie");
      String vergleich =
            kategorieKnoten.item(0).getFirstChild().getNodeValue();
      if (vergleich.equals("edv")) {
        NodeList urlKnoten = eintrag.getElementsByTagName("url");
        neuElement = (Element)ausgabebaum.importNode(eintrag, true);
        wurzel.appendChild(neuElement);
      }
    }
    System.out.println
            (ausgabebaum.getElementsByTagName("adressen").item(0));

    String serial = "<?xml version='1.0' encoding='iso-8859-1'?>\n";
    serial += ausgabebaum.getElementsByTagName("adressen").item(0);

    try {
      FileWriter Datenstrom = new FileWriter("teil.xml");
      BufferedWriter ausgabe = new BufferedWriter(Datenstrom);
      ausgabe.write(serial);
      ausgabe.flush();
      ausgabe.close();
    }
    catch (IOException e) {
      System.out.println(e);
    }
  }
}
```

Zu Beginn werden zwei `document`-Objekte erzeugt: Das `document`-Objekt `xmlbaum` enthält die ursprünglichen XML-Elemente. Die `document`-Methode `appendChild()` erzeugt für den ursprünglich leeren XML-Baum `ausgabebaum` das Wurzelelement. Alle Knoten, dessen Kategorie-Element den Wert `edv` hat, werden mit der `document`-Methode `importNode()` als neues Element für den Ausgabebaum erzeugt. Diese neuen Elemente werden schließlich mit der Element-Methode `appendChild()` dem Wurzelelement als Kindelemente hinzugefügt.

Bevor wir den neuen XML-Baum in eine Textdatei speichern, geben wir ihn – zur Kontrolle - mit einer einfachen `System.out.println()` - Anweisung auf der Konsole aus:

```
<adressen>
      <eintrag id="0">
                <kategorie>edv</kategorie>
                <url>http://www.w3.org</url>
                <notiz>W3-Konsortium</notiz>
                <notiz>Technische Referenz</notiz>
      </eintrag>
      <eintrag id="4">
                <kategorie>edv</kategorie>
                <url>http://www.suse.de</url>
                <notiz>SuSE, Linux</notiz>
                <notiz>Treiber-Datenbank</notiz>
      </eintrag>
      <eintrag id="5">
      ...
</adressen>
```

Schließlich schreiben wir die gesamte Ausgabe in eine Stringvariable und speichern den gesamten Inhalt in der Textdatei `teil.xml`. Dabei erhält diese Datei als ersten Eintrag die notwendige XML-Deklaration.

```
<?xml version='1.0' encoding='iso-8859-1'>
<adressen>
      <eintrag id="0">
                <kategorie>edv</kategorie>
                <url>http://www.w3.org</url>
                <notiz>W3-Konsortium</notiz>
                <notiz>Technische Referenz</notiz>
      </eintrag>
      <eintrag id="4">
                <kategorie>edv</kategorie>
                <url>http://www.suse.de</url>
                <notiz>SuSE, Linux</notiz>
                <notiz>Treiber-Datenbank</notiz>
      </eintrag>
      ....
</adressen>
```

Eine weitere Anwendung für das Erstellen neuer XML-Dateien liegt vor, wenn die Elemente nach einem bestimmten Kriterium umgeordnet, z.B. alphabetisch sortiert werden sollen. Eine andere interessante Anwendung besteht darin, aus einem bestehenden XML-Dokument Dokumente für verschiedene Ausgaben zu erzeugen – etwa für die Ausgabe in einem Browser (XHTML) oder für die Druckausgabe (XSL-FO).

5. Aufgaben, Ausblick

1. Aus einer vorgegebenen XML-Datei ist eine neue XML-Datei zu erzeugen, in der die Einträge alphabetisch sortiert sind.
2. Der Inhalt einer XML-Datei soll mit dem Swing-Objekt `JTree` dargestellt werden.
3. Die Bedeutung von XSLT (*Extended Stylesheet Language Transformations*) zum Erzeugen neuer Dokumentenbäume sollte jedenfalls mit den Möglichkeiten eines XML-Parsers verglichen werden.
4. Bei der Ausgabe eines XML-Dokumentenbaumes in eine Textdatei müssen alle Elemente des Baumes der Reihe nach geschrieben werden. Man spricht in diesem Zusammenhang von der „Serialisierung von Daten“. Dieses Konzept sollte anhand anderer Tools erweitert werden.

6. Literatur, Weblinks

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

[2] http://www.w3.org/TR/REC_DOM-Level-2 (Vollständige Spezifikation des W3C-Konsortiums)

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

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

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

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

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

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

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