JAVA und SAX

XML-Dokumente verarbeiten

Alfred Nussbaumer

In PCNEWS-87 ([1]) wurde beschrieben, wie XML-Dokumente an Hand des DOM (*Document Object Model*) und entsprechenden JAVA-Klassen verarbeitet werden können. Ein grundlegend anderes Konzept wird vom SAX (*Simple API for XML Parsing*) realisiert. Obwohl SAX im Gegensatz zum DOM in keiner offiziellen Standardisierung durch das W3C vorliegt, ist es weit verbreitet. Es wird derzeit in der Version 2.0 durch zahlreiche Java-Klassen unterstützt. Diese sind ab der Version 1.4 in JAVA enthalten (siehe [3]).

1. Grundlagen

Ein SAX-Parser lädt nicht das ganze Dokument auf einmal, sondern arbeitet es in einzelnen Teilen, in so genannten SAX-Ereignissen, ab. Beispielsweise unterscheidet SAX zwischen dem Auftreten eines Start-Tags (z.B. `<eintrag>`) und eines Ende-Tags (z.B. `</eintrag>`). SAX arbeitet in bestimmten Anwendungen wesentlich schneller als DOM und kann auch große XML-Dokumente verarbeiten.

Die grundlegenden Klassen sind in den Packages `javax.xml.parsers`, `org.xml.sax` und `org.xml.sax.helpers` enthalten. Für das folgende Beispiel (und für die weiteren) verwenden wir die einfache 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>
```

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)>
```

Wir parsen die XML-Datei mit folgenden wenigen Programmzeilen (`Sax1.java`):

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

public class Sax1 extends DefaultHandler {

  static Sax1 sax;

  public static void main (String args[]) throws Exception {
    sax = new Sax1();
    sax.parsen();
  }
  public void parsen() throws Exception {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();
    reader.setContentHandler(sax);
    reader.parse("weblinks.xml");
  }
  public void endDocument() throws SAXException {
    System.out.println("weblinks.xml wurde geparst");
  }
}
```

Innerhalb der Methode `parsen()` wird zunächst die neue Instanz `factory` der Klasse `SAXParserFactory` erzeugt. Sie stellt den `XMLParser` `parser`, und dieser den `XMLReader` `reader` zur Verfügung. Hier bezieht sich die Methode `setContentHandler()` auf das aktuelle Klassenexemplar (`sax`); die Methode `parse()` arbeitet das angegebene XML-Dokument (`weblinks.xml`) ab.

Die Klasse `Sax1` erweitert die Klasse `DefaultHandler` aus dem Package `org.xml.sax.helpers` und überschreibt die Methode `endDocument()`: Auf diese Weise wird nur am Ende des Parsens, also wenn das Ende des XML-Dokuments erreicht wurde, eine Mitteilung auf die Standardausgabe geschrieben.

Findet der Parser das XML-Dokument und die angegebene DTD im aktuellen Verzeichnis, erhalten wir lediglich die Ausgabe

```
weblinks.xml wurde geparst
```

Dies scheint wenig spektakulär. In der Tat erwarten wir lediglich Fehlermeldungen, wenn beispielsweise die angegebene XML-Datei oder die DTD-Datei nicht gefunden wird. Im ersten Fall erhalten wir die *„FileNotFoundException"* - Fehlermeldung vom `FileInputStream`, im zweiten Fall liefert der XML-Parser (crimson) die gleiche Exception.

Wesentlich wichtiger ist jedoch die Tatsache, dass der Parser die verwendete XML-Datei daraufhin überprüft, ob sie „wohlgeformt" ist. Ein XML-Dokument hat diese Eigenschaft, wenn u. A. gilt (eine genaue Aufzählung aller Merkmale für Wohlgeformtheit finden Sie beispielsweise in [6]):

1. Das XML-Dokument besitzt genau ein Wurzelelement (Dokumentelement). Alle anderen Elemente hängen von diesem Wurzelelement ab.
2. Alle Elemente sind korrekt geschachtelt; sie bilden eine Baumstruktur mit dem Wurzelelement als Wurzel des Baumes.
3. Jedes Element enthält ein Start-Tag und ein Ende-Tag.
4. Das Start-Tag kann eindeutig benannte Attribute enthalten.
5. Alle Attribute bilden Attributname-Attributwert-Paare.

Sind obige Regeln zur Wohlgeformtheit nicht erfüllt, so liefert der SAX-Parser Fehlermeldungen wie folgende:

```
Exception in thread "main" org.xml.sax.SAXParseException:
"</kategorie>" zum Abschließen des Elements auf Zeile 11 erwartet
```

In diesem Fall wurde offensichtlich die Regel zur korrekten Schachtelung der XML-Elemente verletzt.

```
Exception in thread "main" org.xml.sax.SAXParseException: Nächstes
Zeichen muss "=" nach Attributname "name" sein.
```

Hier wurde ein Attributname ohne Wert angegeben.

Der SAX-Parser kann eine XML-Datei neben dem Test auf Wohlgeformtheit auch auf ihre „Gültigkeit" hin überprüfen: Eine XML-Datei heißt gültig, wenn sie eine DTD (*Document Type Definition*) verwendet. Soll der Parser die XML-Datei gegen die DTD validieren, so ruft man die `SAXParserFactory`-Methode `setValidating()` mit dem Wert `true` auf. Damit wird beispielsweise überprüft, ob jedes `<eintrag>`-Element genau ein `<kategorie>`-Element, genau ein `<url>`-Element, ein oder mehrere `<notiz>`-Elemente und kein oder ein `<datum>`-Element enthält. Im folgenden Programmtext wurde anstelle der statischen Klassenvariable `sax` die Referenz `this` verwendet, die sich auf das aktuelle Klassenexemplar der Methode `parsen()`, also auf `sax` bezieht.

```
...
public class Sax1 extends DefaultHandler {

  public static void main (String args[]) throws Exception {
    Sax1 sax = new Sax1();
    sax.parsen();
  }
```

```
  public void parsen() throws Exception {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setValidating(true);
    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();
    reader.setContentHandler(this);
    reader.parse("weblinks.xml");
  }
  public void endDocument() throws SAXException {
    System.out.println("weblinks.xml wurde geparst");
  }
}
```

Das Interface `ErrorHandler` enthält drei Methoden für verschieden gravierende Fehler: `warning()`, `error()` und `fatalError()`. Durch Überschreiben dieser Methoden können spezifische Fehlermeldungen bzw. eine individuelle Ausgabe der Fehlermeldungen auf einem bestimmten Medium erreicht werden. Voraussetzung ist jedenfalls, dass die Klasse `SAXParseException` importiert und eine neue Instanz von `ErrorHandler` erzeugt wurde:

```
...
import org.xml.sax.SAXParseException;

  public void parsen() throws Exception {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();
    reader.setContentHandler(this);
    reader.setErrorHandler(this);
    reader.parse("weblinks.xml");
  }
```

Nun können die Fehler-Methoden wunschgemäß überschrieben werden, z.B:

```
...
  public void warning(SAXParseException e) {
    System.out.println("WARNUNG: " + e);
  }

  public void error(SAXParseException e) {
     System.out.println("ERROR!" + e);
  }

  public void fatalError(SAXParseException e) {
    System.out.println("Schwerwiegender Fehler!!!" + e);
  }
```

Die `SAXParseExecption`-Methoden `getLineNumber()` und `getMessage()` geben Auskunft über den Ort und über die Art des Parsefehlers; `getSystemId()` liefert den Pfad und den Dateinamen der geparsten XML-Datei. Eine detaillierte Beschreibung zu den in diesem Abschnitt vorgestellten Klassen, Interfaces und Methoden wird in der JAVA-Dokumentation (*[3]*) gegeben. Zusätzlich kann das Tutorial zu WebServices von SUN empfohlen werden (*[4]*).

2. Elemente und Attribute anzeigen

Im Beispiel `Sax2.java` sollen die Zeichenkettenwerte der Textelemente und Attributwerte ausgegeben werden. Dazu erweitern wir wieder die Klasse `DefaultHandler` und überschreiben einige Methoden, die sich auf bestimmte SAX-Ereignisse beziehen:

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;

public class Sax2 extends DefaultHandler {

  public static void main (String args[]) throws Exception {
    Sax2 sax = new Sax2();
    sax.parsen();
  }

  public void parsen() throws Exception {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();
    reader.setContentHandler(this);
    reader.parse("weblinks.xml");
  }
  public void startDocument() {
  }
  public void startElement(String nsURI, String localName,
      String qName, Attributes atts) throws SAXException {
    System.out.print("Element " + qName + ": ");
    if (atts.getValue(0) != null)
      System.out.println("Attribut id : " +
          atts.getValue(0));
  }
  public void characters(char[] ch, int start, int length)
    throws SAXException {
   String textelement = new String(ch, start, length);
   System.out.print(textelement);
  }
  public void endElement(String nsURI, String localName,
    String qName) throws SAXException {
    System.out.println();
  }

  public void endDocument() throws SAXException {
    System.out.println("weblinks.xml wurde geparst");
  }
}
```

Die im obigen Beispiel auftretenden SAX-Ereignisse sind der Beginn des Dokuments, das Auftreten eines Start-Tags, ein Textinhalt, das Auftreten eines Ende-Tags und das Erreichen des Dokumentendes. Aus diesem Grund müssen die Methoden `startDocument()`, `startElement()`, `characters()`, `endElement()` und `endDocument()` entsprechend überschrieben werden. Die Methode `startElement()` gibt die Mitteilung `„Element:"`, den Namen des Elementes und ggf. das angegebene Attribut mit seinem Attributwert auf der Konsole aus. Die Methode `characters()` liefert den Zeicheninhalt eines Elementes in Form eines Arrays, der in Form einer Zeichenkette ausgegeben wird. Die Methode `endElement()` liefert nur einen Zeilenvorschub auf der Konsole:

```
Element weblinks: Element eintrag: Attribut id : 0
Element kategorie: edv
Element url: http://www.w3.org
Element notiz: W3-Konsortium
Element notiz: Technische Referenz

Element eintrag: Attribut id : 1
Element kategorie: phy
Element url: http://www.cern.ch
Element notiz: Europaeisches Kernforschungszentrum
Element notiz: Aktuelles zur Hochenergiephysik
Element notiz: Materialien zur Elementarteilchenphysik

Element eintrag: Attribut id : 2
Element kategorie: ast
Element url: http://www.nasa.gov
Element notiz: Amerikanische Weltraumfahrtbehoerde
Element notiz: Aktuelle Raumfahrtprojekte
Element notiz: Historische Daten
...
```

Die `Attributes`-Methode `getValue(index)` liefert den Attributwert zum Attribut mit dem angegebenen Index. Im Beispiel tritt nur ein Attribut auf, es hat den Index 0.

3. Bestimmte Elemente auswählen

Im letzten Beispiel `Sax3.java` sollen Elemente anhand ihres Namens und Textwertes ausgewählt und ausgegeben werden. Konkret sollen die Webadressen und alle Notizen zu einer bestimmten Kategorie ausgewählt und ausgegeben werden. Der Textwert des Elementes `<kategorie>` wird auf der Kommandozeile als erstes Argument übergeben. Falls dieser Wert fehlt, wird im Hauptprogramm eine entsprechende Fehlermeldung ausgegeben und das Programm abgebrochen.

Da die `DefaultHandler`-Methoden `startElement()` und `characters()` grundsätzlich unabhängig voneinander aufgerufen werden, müssen wir das Zusammenspiel zwischen diesen beiden Methoden über zusätzliche Klassenvariablen regeln. Im Beispiel werden dafür die Variablen `kategorie_flag` und `ausgabe_flag` verwendet. Zusätzlich soll die Anzahl aller passenden Elemente ermittelt werden: Die Variable treffer wird beim SAX-Ereignis „Beginn des Dokumentes" auf 0 gesetzt und jedesmal inkrementiert, wenn der Textinhalt des `<kategorie>`-Elementes (und nur dieser) mit der ausgewählten Bezeichnung übereinstimmt.

```
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;

public class Sax3 extends DefaultHandler {

  public String kategorie;
  public int treffer;
  public int kategorie_flag;
  public int ausgabe_flag;
```

```
  public static void main (String args[]) throws Exception {
    Sax3 sax = new Sax3();
    if (args.length == 0) {
      System.out.println("Kategorie angeben! ");
      System.out.println("z.B.: > java Sax3 kat");
      System.exit(0);
    }
    else {
      sax.kategorie = args[0];
      sax.parsen();
    }
  }
  public void parsen() throws Exception {
    XMLReader reader =
      SAXParserFactory.newInstance().newSAXParser().getXMLReader();
    reader.setContentHandler(this);
    reader.parse("weblinks.xml");
  }
  public void startDocument() {
    treffer = 0;
  }
  public void startElement(String nsURI, String localName,
    String qName, Attributes atts) throws SAXException {
    if (qName.equals("kategorie")) {
      kategorie_flag = 1;
      ausgabe_flag = 0;
    }
  }
  public void characters(char[] ch, int start, int length)
    throws SAXException {
    String textelement = new String(ch, start, length);
    if (kategorie_flag == 1)
      if (textelement.equals(kategorie)) {
        ausgabe_flag = 1;
        treffer++;
      }
    if (ausgabe_flag == 1) {
      System.out.println(textelement);
    }
  }
  public void endElement (String nsURI, String localName,
    String qName) throws SAXException {
    if (qName.equals("kategorie")) kategorie_flag = 0;
  }
  public void endDocument() throws SAXException {
    System.out.println("\nweblinks.xml wurde geparst - " +
      treffer + " Treffer");
  }
}
```

Was passiert bei den verschiedenen SAX-Ereignissen? Beim Ereignis „`startDocument`“, also wenn der Parser mit dem Lesen der XML-Datei beginnt, wird der Wert der Zählvariablen treffer auf null gesetzt. Wird das Start-Tag des `<kategorie>`-Elementes gefunden, so wird die Variable `kategorie_flag` auf `1` gesetzt. Damit kann beim Auftreten des Textelementes (also in der Methode `characters()`) überprüft werden, ob der Textwert der ursprünglich angegebenen Kategorie-Bezeichnung entspricht. Im Fall einer Übereinstimmung erhält die Variable `ausgabe_flag` den Wert `1`, und alle weiteren Textelemente werden ausgegeben bis das nächste Mal das Start-Tag zum Element `<kategorie>` auftritt. In diesem Fall wird die Variable `ausgabe_flag` wieder auf `0` zurückgesetzt.

Um zu vermeiden, dass der Textinhalt irgendeines anderen Elementes mit der Kategoriebezeichnung verglichen wird, setzen wir beim Auftreten des Ende-Tags des `<kategorie>`-Elementes den Wert der Variablen `kategorie_flag` auf `0` zurück. Somit wird innerhalb eines `<eintrag>`-Elementes nur einmal auf den Textinhalt des `<kategorie>`-Elements geprüft.

Wenden wir das Beispiel `Sax3.java` auf die eingangs genannte XML-Datei an und wählen wir als Kategoriebezeichnung beispielsweise `„phy"` aus, so erhalten wir einen Treffer:

```
nus@ice:~/java/xml/sax> java Sax3 phy
phy
http://www.cern.ch
Europaeisches Kernforschungszentrum
Aktuelles zur Hochenergiephysik
Materialien zur Elementarteilchenphysik

weblinks.xml wurde geparst - 1 Treffer
```

Die gleiche Aufgabenstellung lässt sich auch lösen, indem man nur eine Variable `ausgabe_flag` verwendet. Dieser Variablen weisen wir verschiedene Werte zu um die verschiedenen Ereignisse von einander zu unterscheiden. Im folgenden Ausschnitt erhält sie den Wert `1`, wenn das Start-Tag des Elementes `<kategorie>` auftritt. Entspricht das folgende Textelement der gewünschten Kategorie, so wird die Treffervariable inkrementiert und die Variable `ausgabe_flag` auf den Wert `2` gesetzt. So lange sie genau diesen Wert hat, werden alle weiteren Textelemente (also die Adresse und alle zugehörigen Notizen) ausgegeben. Zuletzt wird das Ereignis überprüft, das beim Ende-Tag eintritt: Beim Ende-Tag des `<eintrag>`-Elementes setzen wir `ausgabe_flag` in jeden Fall auf Null zurück, damit nicht die Textelemente des nächsten Eintrages ausgegeben werden. Tritt das Endetag von `<kategorie>` auf, so setzen wir die Variable `ausgabe_flag` nur dann auf Null, wenn nicht gleichzeitig die gewünschte Kategorie auftritt – denn in diesem Fall sollen ja alle zugehörigen Textelemente ausgegeben werden:

```
  public void startElement(String nsURI, String localName,
    String qName, Attributes atts) throws SAXException {
    if (qName.equals("kategorie")) {
      ausgabe_flag = 1;
    }
  }
  public void characters(char[] ch, int start, int length)
    throws SAXException {
    String textelement = new String(ch, start, length);
    if (ausgabe_flag == 1)
      if (textelement.equals(kategorie)) {
        ausgabe_flag = 2;
        treffer++;
      }
    if (ausgabe_flag == 2) {
      System.out.println(textelement);
    }
  }
  public void endElement (String nsURI, String localName,
    String qName) throws SAXException {
    if (qName.equals("eintrag")) ausgabe_flag = 0;
    if (qName.equals("kategorie") && (ausgabe_flag != 2))
      ausgabe_flag = 0;
  }
```

Das Verfahren, die Ausgabe von Textinhalten mittels einer Variablen zu steuern, kann auch auf kompliziertere Situationen übertragen werden. In diesem Fall ordnet man bestimmten Ereignissen gewisse voneinander verschiedene Zahlenwerte zu.

4. Aufgaben, Ausblick

1. Ein XML-Dokument mit mehreren Ebenen soll hinsichtlich bestimmter Elemente und in Hinblick auf einen bestimmten Inhalt durchsucht werden. Der Elementname und der gewünschte Inhalt sollen mit Hilfe zweier Argumente angegeben werden.
2. Die Ein- und Ausgaben in den vorangegangenen Beispielen sind mit Hilfe von GUI-Objekten zu realisieren.
3. Für die DOM- und SAX-Schnittstelle gibt es die spezielle JAVA-Entwicklung JDOM. Außerdem lässt JDOM die Transformation von XML-Dateien mittels XSLT zu. Mit den in JDOM zur Verfügung gestellten Klassen lassen sich JAVA-Programme zur Verarbeitung von XML-Daten leicht formulieren.

5. Literatur, Weblinks

[1] „JAVA und DOM“, PCNEWS-87 April 2004, S. 31

[2] *http://www.w3.org/TR/REC-xml* (W3C-Empfehlung zu XML, Version 1.0)

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

[4] *http://java.sun.com/webservices/docs/1.3/tutorial/doc/index.html* (Dokumentation zu WebServices), als PDF-Dokument unter der Adresse *http://java.sun.com/webservices/docs/1.3/tutorial/doc/JavaWSTutorial.pdf*

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

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

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

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

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

[10] *http://nus.lugsp.at/informatik/wpf/JAVA/index.php* (Unterrichtsbeispiele zum Programmieren mit JAVA)