7 XML und Java

Mitte der 90er Jahre wurde Java vorgestellt. Zur gleichen Zeit gab es erste Vorschläge für XML. Obwohl kein unmittelbarer Zusammenhang zwischen diesen beiden Entwicklungen besteht, kann man doch sagen, dass beide Technologien sich hervorragend ergänzen:

- Java ist die plattform-unabhängige **Programmier**sprache.
- XML ist die plattform-unabhängige **Daten**beschreibungssprache.

Dieses Kapitel beschreibt zunächst anhand von Beispielen, was XML eigentlich ist und wozu es eingesetzt werden kann - unabhängig von einer konkreten Programmiersprache und damit auch unabhängig von Java. Es ist nicht Ziel dieses Buches, eine Einführung in XML oder XSLT zu geben, sondern es soll gezeigt werden, wie Java - Anwendungen vom Einsatz der Beschreibungssprache XML profitieren können.

Deshalb wird sich der größte Teil des Kapitels mit dem Java-Standard **JAXP** (= Java API for XML-Processing) beschäftigen. Es wird gezeigt, wie XML-Dokumente mit diesem API

- erstellt
- geprüft
- analysiert
- transformiert und
- generiert

werden können.

7.1 Grundlagen XML

7.1.1 Einführung XML-Dokumente

Was ist XML?

XML wurde 1998 vom W3C ("World Wide Web Consortium") als Standard verabschiedet. Diese Meta-Sprache dient dazu, strukturierte Daten

- universell zu **beschreiben** (unabhängig von der Verwendung)
- **auszutauschen** (portabel über Plattform- und Programmiersprachen-Grenzen)
- zu **transformieren** (in beliebige Ausgabeformate).

Wie ist eine XML-Datei aufgebaut?

XML-Dateien enthalten beides, die aktuellen Daten und die Beschreibung der Struktur.

Ähnlich wie bei HTML wird die Struktur durch sog. „**Tags**“ beschrieben. Tags sind Bezeichner für das eigentliche Textelement – dadurch können die Nutzdaten identifiziert werden. *Tags* sind in spitzen Klammern eingefasst, z. B.

<person>Erwin Merker</person>.

Der Start-*Tag* ist <person>, dann folgen die Nutzdaten, zum Schluss das Ende-*Tag* </person>. In XML beschreiben *Tags* die Bedeutung der von ihnen eingeschlossenen Daten, während in HTML die Art der Formatierung der Daten über *Tags* gesteuert wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<application
   xmlns="http://java.sun.com/xml/ns/j2ee">

  <display-name xml:lang="de">
   EntityApp
  </display-name>

  <module>
    <java>app-client-ic.jar</java>
  </module>

  <module>
    <ejb>ejb-jar-ic.jar</ejb>
  </module>

</application>
```

Abb. 7.1.1: Beispiel einer XML-Datei

Es gibt in XML keine fest vorgegebenen *Tag*-Namen, sie sind frei wählbar. Die Konsequenz ist natürlich, dass sie nichts verraten über die Bedeutung eines XML-Elements und somit ein Empfänger nicht weiss, wie die einzelnen Text-Elemente zu interpretieren sind. Zum Beispiel kann ein Browser mit einer XML-Datei nicht viel anfangen, weil die *Tags* keinerlei Hinweise enthalten, **wie** die Daten zu formatieren oder anzuzeigen sind.

Also: Für die Interpretation der *Tag*-Semantik ist die jeweilige individuelle Anwendung zuständig, d.h. der Nutzer einer XML-File muss die *Tag*-Bedeutung kennen. Wenn der Ersteller einer XML-Datei einen *Tag* <name>, der Empfänger für denselben Inhalt den *Tag* <vorname> erwartet, führt dies wahrscheinlich zu Kommunikationsproblemen und Missverständnissen.

Was kann man mit einem XML-Dokument machen?

Das Ziel der *Tags* ist es also,

- den Aufbau und die inhaltliche Struktur eines Dokuments zu beschreiben und
- die einzelnen Elemente mit Namen zu versehen.

Generell kann XML eingesetzt werden für die Beschreibung von

- einzelnen **Daten** (z.B. Parameter, die ausgetauscht werden) und von
- kompletten **Dokumenten** (z.B. Texte, die strukturiert sind oder Dateien, die einen hierarchischen Aufbau haben).

Für die Verarbeitung eines XML-Dokuments ist es erforderlich, dass die Datei zunächst in den Hauptspeicher eingelesen, geprüft und analysiert wird. Und weil dies auf immer die gleiche Art und Weise erfolgt, hat man die Verfahren dafür formalisiert und standardisiert.

Einlesen von XML-Dateien

Für das Einlesen - und Prüfen - von XML-Dateien gibt es zwei Standardverfahren, die unabhängig von Programmmiersprachen definiert worden sind. Sie legen fest, wie der Inhalt des Dokuments durchsucht wird nach *Tag*-Elementen, damit die Nutzdaten dann entsprechend herausgelesen und dem Anwendungsprogramm zusammen mit Informationen über die Schachtelungsebene übergeben werden können. Die beiden Verfahren sind:

- DOM-Standard
  Vor der Verarbeitung wird die komplette Datei in den Hauptspeicher gelesen und eine Baumstruktur aus den Elementen erstellt. Dieser Baum (*tree*) kann dann von der Anwendung nach festen Regeln (mit vorgegeben DOM- APIs) durchlaufen und geändert werden. Dieses Verfahren wird auch "**tree-based**" genannt. Wichtige Fähigkeit: Ein wahlfreier Zugriff auf die Datenelemente ist möglich.

- SAX-Standard
  Die Daten werden nacheinander eingelesen und sofort der zu einem Einzelelement gehörende Programmcode für die Verarbeitung aufgerufen (abhängig von der Art des gerade gelesenen *Tags*, z.B. Elementstart oder Elementende). Dieses Verfahren wird auch "**event-gesteuert**" genannt. Wichtige Einschränkung: Es ist nur sequentieller Zugriff auf den Dokumenten-Inhalt möglich.

Weil diese APIs programmiersprachen-unabhängig sind, müssen sie für die jeweilige Programmiersprache konkretisiert und in Produkte (= "**Parser**") gegossen werden.

Prüfen von XML-Dateien

Wichtig ist darüber hinaus, dass beim Einlesen auch eine maschinelle Überprüfung der XML-Dokumente möglich ist. Dabei unterscheidet man::

- Überprüfung, ob gegen Syntaxregeln der XML-Spezifikationen verstoßen wurde. Man sagt: es wird geprüft, ob das Dokument "**wohlgeformt**" ist.
- Überprüfung, ob alle logischen Anforderungen erfüllt sind. Man sagt: es wird geprüft, ob das Dokument "**gültig" ("valide**") ist.

Die logischen Anforderungen für die Prüfung auf Validität eines XML-Dokument werden in einer separaten Datei (DTD oder Schema-Files) festgelegt. Im Wesentlichen stehen dort die Definitionen der einzelnen *Tags:* Welche Tags gibt es? Welche müssen und welche können vorhanden sein? Wie ist die Reihenfolge? Welche Werte sind erlaubt? Von welchem Datentyp muss das Element sein? usw.

Was ist ein Parser?

Der Parser ist das standardisierte Kernmodul einer jeden XML-Anwendung. Um ein XML-Dokument zu verarbeiten, wird das Anwendungsprogramm die XML-Datei dem Parser übergeben, damit dieser das Dokument für die Application

- einliest und seine Struktur analysiert und außerdem
- überprüft, ob der XML-Code wohlgeformt und gültig ("valide)" ist.

Die weitere Vorgehensweise in der Applikation hängt dann von dem gewählten Verfahren ab:

- Bei dem DOM-Modell bekommt die Anwendung als Ergebnis die komplette Datei in Baumstruktur zur Verfügung gestellt.
- Bei dem SAX-Modell generiert der Parser fortlaufend Events, die zum Aufruf von Methoden führen, die dann als Teil der eigentlichen Anwendungen die Verarbeitung durchführen.

Die Prüfung auf Validität ist benutzer-optional. Sie wird

- nur von sogenannten "validierenden" Parsern durchgeführt und
- dies auch nur auf ausdrücklichen Wunsch.

Parser werden in Verarbeitungsprogramme als "plug-in" oder als Unterprogramm eingebunden.

Die Implementierung der Parser-Funktionalität erfolgt durch beliebige Produkte. Ein Beispiel für eine Referenz-Implementierung ist XERCES von Apache. Diese Implementierung gibt es in mehreren Programmiersprachen, auch in Java. Das Produkt unterstützt das DOM- und das SAX-Verfahren.

Transformieren von XML

Ein weiterer Standard besteht für das Transformieren einer XML-Datei. Damit wird ein Vorgang bezeichnet, der ein XML-Dokument in ein anderes Format überführt oder der die Struktur der Datei verändert. Dies kann mit Hilfe der Sprache XSL geschehen – und diese Sprache ist auch programmiersprachen-unabhängig.

Für die konkrete Arbeit wird ein "XSLT-Prozessor" benötigt. Das ist ein Programm, das die eigentliche Transformation anhand der XSL-Beschreibung durchführt. So gibt es für Java die Referenz-Implementierung "XALAN" von Apache.

XML versus HTML

- HTML stellt eine feste Menge **vordefinierte**r Elemente bereit. In XML sind die „*Tag*"-Namen nicht vorgegeben, sondern sie können **selbst definiert** werden, z.B. <person> oder <alter>. Das gilt

auch für Attribute. Dadurch ist die Anzahl der Tags in XML nicht begrenzt (daher auch die Bezeichnung "*ex**tensible markup language*").

- In HTML sind die *Tag*-Namen ausgerichtet auf die Interpretation durch den Browser. In XML ist die Strukturbeschreibung nicht ausgerichtet auf die *Darstellung* der Daten durch **ein** bestimmtes Programm (=Browser), sondern auf die universell gültige Beschreibung des Inhalts, unabhängig von einer konkreten Aufgabenstellung oder gar von einem speziellen Programm. Eine XML-Datei soll inhaltliche Informationen über die beschriebenen Daten für beliebige Software verständlich darstellen.
- Eine Interpretation der Daten ist für den Empfänger der XML-Datei allerdings nur möglich, wenn er die semantische Bedeutung der *Tag*-Namen kennt. Hier können weitere Dateien Hilfe leisten: DTD-File, XML Schema oder XSL-Datei (dazu später mehr).
- Formale Unterschiede:
  - XML ist case-sensitive.
  - XML hat strengere Syntaxregeln: „nicht-wellformed“ Dokumente verursachen Abbruchfehler; es gibt keine Fehlertoleranz wie bei HTML.
  - Attribute stehen bei XML in Hochkomma.
  - Attribute *müssen* bei XML einen Wert haben.
- XML ist eine strukturierende Sprache, die keine Layout-Vorschriften enthält.

7.1.2 Installationsvoraussetzungen für XML

API für XML

Die dringende Empfehlung für das Arbeiten mit allen XML-Beispielen: Installation der J2EE SDK 1.4 Plattform. Diese Java2 Plattform enthält das JAXP API. Frühere Versionen sind unvollständig und erfordern zusätzliche Installationsarbeiten.

Die Java 2 Standard Edition enthält folgende **APIs** für das Arbeiten mit XML:

- *javax.xml.parsers* (für die Factory)
- *javax.xml.transform* (für dom, sax und streams).

Wie bei APIs üblich, können beliebige Produkte eingesetzt werden, sofern diese die APIs auch implementiert haben. Ein Wechsel des Produktes oder des Herstellers bedeutet dann keinerlei Änderungen der Anwendungs-Sourcen.

Produkte

Benötigt wird die J2EE-Version vom 17.3.2004 oder später (siehe beiliegende CD). In diesem Paket sind u.a. alle **Referenz-Implementierungen** für das JAXP-API enthalten, so dass alle Arten von Arbeiten mit XML-Files möglich sind (Prüfen, Parsen, Transformieren...):

Parser für DOM (XERCES)

Parser für SAX (XERCES)

Transformer (XALAN).

Die JAR-Files mit diesen Produkten stehen in folgenden Ordnern:

*X:\Sun\lib\endorsed**xercesImpl.jar**;*

*X:\Sun\lib\endorsed**xalan.jar**;*

*X:\Sun\lib\endorsed**dom.jar**;*

Für einige Beispiele werden zusätzliche Produkte (in Form von JAR-Files) notwendig. Wenn hierfür zusätzlich auch der Classpath angepasst werden muss, wird dies in den entsprechenden Abschnitten erläutert.

Classpath-Hinweise

Bei Arbeiten mit XML-APIs ist der richtige CLASSPATH ein nicht zu unterschätzendes Problem. Das liegt daran, dass die gesamte XML-Technologie ein schnell wachsendes und sich ständig änderndes Thema ist. Einige APIs gehören noch nicht zum Java-Standard. Außerdem gibt es viele Implementierungen, die sich stark unterscheiden, d.h. die Parser-Implementierungen können in unterschiedlichen Versionen mitgeliefert sein, z.B. bei J2EE, bei J2SE 1.4 oder beim JWSDP. Deshalb die Empfehlung: bei Problemen immer zunächst den Classpath prüfen.

7.1.3 Bedeutung von XML

Was ist ein XML-Dokument?

Ein XML-Dokument ist eine Text-File. Sie enthält:

- Verarbeitungs-Instruktionen = Informationen für den XML-Prozessor
- Elemente = Basis für die Beschreibung der Verschachtelung
- Attribute = Geben zusätzliche Informationen für die Elemente
- Daten = eingeschlossen in *Tag*-Namen.

Die **Tags** sind case-sensitiv, Groß-/Kleinschreibung muss beachtet werden. Die *Tag*-Namen sind nicht vordefiniert, sondern frei wählbar. Für das Arbeiten mit XML-Dokumenten gibt es eine Fülle von **neuen Technologien**, z.B.

- DTD/XSD-Schema (= beschreiben)
- DOM/SAX (= einlesen/prüfen)
- XSL (= darstellen/transformieren)
- XLink (= verbinden)
- XPath (= zugreifen)
- XML-Query (= abfragen)
- Encoder (= maschinell erstellen) usw.

Alle Technologien sind zunächst einmal programmiersprachen-unabhängig. Im praktischen Einsatz sind vor allem die beiden Plattformen:

- Java-Technologien und die
- Microsoft-Plattform .NET.

Einsatzmöglichkeiten für XML

- XML ist ideal für langfristige Daten-Ablage.
- XML ist besonders geeignet für Dokumente mit hierarchisch geschachtelten Strukturen (SQL dagegen ist besser geeignet für Daten mit festen, vordefinierten Datentypen).
- Transformation in jedes gewünschte Ausgabeformat ist möglich (WML, HTML, PDF oder auch neue Formate, die heute noch nicht bekannt sind), auch für unterschiedliche Medien (Papier, Bildschirm, DB). So könnten Servlets oder JSPs die Ausgabedaten grundsätzlich in XML erstellen, und "vor Ort" können diese beliebig ausgewertet werden, z.B. durch einen Browser mit eingebautem Transformations-Modul in HTML oder von einem Handy in WML.
- XML ist gut geeignet, um über das Web
  - Daten auszutauschen mit anderen **Applikationen** (plattform- und sprachen-unabhängig)
  - Daten auszutauschen mit anderen **Unternehme**n (e-Commerce, Alternative zu EDIFACT).
- Stark strukturierte Informationen können beliebig sortiert, gefiltert oder präsentiert werden.
- Suchmaschinen / Agenten können die Dokumente auf intelligente Art durchsuchen nach *Tag*-Namen (Voraussetzung: *Tag*-Namen sind sinnvoll gewählt).

- Content-Provider können Dokumente unterschiedlicher Art verwalten und für beliebige Formate anpassen und aufbereiten.

Beispiele für praktische Anwendung

- Configurations-Files sind häufig in XML geschrieben (anstelle von proprietären Lösungen).
- J2EE-Deployment erfolgt via XML-Files (Deployment Descriptoren).
- Struts, ein offener Standard zum Entwickeln von MVC-basierenden Web-Applikationen, verwendet als zentrale Command-File eine XML-Datei.
- Office: Staroffice, Word und Excel speichern die Daten (auch) als XML-Files.
- JSP-Tags auch in XML-Syntax möglich (ab JSP 1.1).
- Messaging-Systeme basieren auch auf XML (= JAXM).
- Entwicklungstools, z.B. ANT und ASANT, verwenden XML in ihren Buildfiles.
- Web-Services basieren ausschließlich auf XML (SOAP-Protokoll).
- Content-Management-Systeme benutzen XML als Standard.

Überblick über die Themen in diesem Kapitel

Auf den folgenden Seiten dieses Buches werden die wichtigsten Technologien an Hand von einfachen Beispielen erläutert:

Validieren der XML-Dokumente

- mit DTD
- mit Schema

Bereitstellen zur Verarbeitung (mit „Parser“)

- mit DOM = Baumstruktur-basierend
- mit SAX = Event-basierend

Transformieren/Präsentieren

- XSLT-Prozessor

Generieren neuer XML-Dokumente

- Encode/Decode

XML-Bewertungen und Einordnung

XML-Vorteile:

- lizenzfrei, gut unterstützt und gut dokumentiert
- reine ASCII-Darstellung:
    - Textfiles leicht zu erstellen
    - kein Compile-Vorgang notwendig
    - kein proprietärer Inhalt
- hersteller-/sprachen-unabhängige Standardisierung
- können auch maschinell **erzeugt** werden mit Tools
- können maschinell interpretiert **und** auch vom Menschen gelesen werden
- können maschinell auf Vollständigkeit und Korrektheit **überprüft** werden
- können mit **Parser** eingelesen und im Speicher strukturiert zur Verfügung gestellt werden (z.B. für das Durchsuchen oder für Updates)
- zukunftssicher, weil beliebig interpretier- und konvertierbar
- medienneutral (Papier-/Bildschirmausgabe).

XML-Nachteile:

- reine ASCII-Darstellung = großer Overhead
- zusätzlicher Platzbedarf auch für die „*Tags*“, d.h. der Anteil des reinen Inhalts gegenüber der Strukturbeschreibung kann sehr gering sein
- geringe Effizienz bei Datenübertragung zwischen Systemen, weil voluminös
- Dokumente müssen geparsed werden, um die Daten zu lesen
- Daten müssen zur Verarbeitung konvertiert werden (in jeweilige Datentypen der Programmiersprache)
- hoher Initialaufwand bei Einführung (für Analyse, DTD, Layout ...).