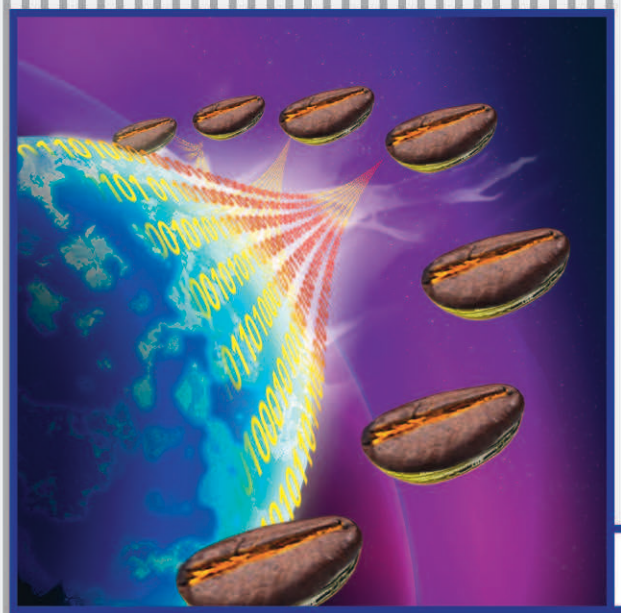

Reinhard Schiedermeier

Programmieren mit Java II

ALWAYS LEARNING
PEARSON

Valide XML-Dokumente

Ein wohlgeformtes XML-Dokument, das einer bestimmten XML-Grammatik folgt, ist valide bezüglich dieser Grammatik. „Validierung“ ist der Test auf Konformität gegenüber einer gegebenen Grammatik. Validität ohne Wohlgeformtheit ist gegenstandslos.

Es gibt eine ganze Reihe von Ansätzen zur Definition von XML-Grammatiken. Zwei Vertreter mit sehr unterschiedlichen Eigenschaften, DTD und XML-Schema, werden hier vorgestellt.

DTD

Schreibweise für XML-Grammatiken

„Document Type Definitions“ (DTDs) sind einer der ersten Formalismen für XML-Grammatiken. DTDs definieren zwar XML-Grammatiken, sind selbst aber keine XML-Dokumente, sondern folgen einer eigenen Syntax.

Eine DTD besteht aus einer Liste von Element- und Attributdefinitionen, deren Reihenfolge keine Rolle spielt. Elemente werden definiert mit

```
<!ELEMENT element content>
```

Inhalt von Elementen

Dieser Ausdruck legt fest, dass alle Elemente mit dem Namen *element* den Inhalt *content* haben. *content* kann einer der folgenden Ausdrücke sein:

`ANY`	Beliebiger Inhalt
`EMPTY`	Kein Inhalt, leeres Element
`(#PCDATA)`	Nur Text
`(#PCDATA` |*element* |...`)*`	Mischung aus Text und Kindelementen in beliebiger Anzahl und Anordnung
`(`*children*`)`	Kindelemente in einer bestimmten Anordnung

Anordnung von Kindelementen

Die folgenden Schreibweisen für Kindelemente *children* legen bestimmte Anordnungen fest. Runde Klammern gruppieren Teilausdrücke und erlauben die Konstruktion geschachtelter Ausdrücke:

element	Ein Element mit dem Namen *element*
children , *children* , ...	Sequenz in fester Reihenfolge
children |*children* |...	Auswahl aus der Liste
children?	Option
children+	Ein- oder mehrmalige Wiederholung
children∗	Beliebige Wiederholung, einschließlich Weglassen

Grenzen der Ausdrucksmächtigkeit

Die Schreibweise ist an reguläre Ausdrücke angelehnt. Sie ist eingängig, hat aber Grenzen. Probleme machen zum Beispiel „alle Elemente aus einer Auswahl in beliebiger Reihenfolge“ und „erst eine beliebige Anzahl Elemente x, dann gleich viele Elemente y“.

Die Definition einer Attributliste bezieht sich auf ein Element und nennt für jedes Attribut Namen, DTD-Typ und Vorgabewert. Die Reihenfolge der einzelnen Attributdefinitionen ist ohne Bedeutung.

```
<!ATTLIST element
   attributename type default
   ...
>
```

Der DTD-Typ *type* begrenzt die möglichen Attributwerte:

Eingrenzung von Attributwerten

CDATA	Beliebiger Text.
ID	Ein Bezeichner, den kein anderes Attribut vom DTD-Typ ID als Wert hat. Bezeichner unterliegen ähnlichen Einschränkungen wie Element- und Attributnamen.
IDREF	Ein Bezeichner, der als Wert eines Attributs vom DTD-Typ ID vorkommt.
IDREFS	Mit Leerzeichen getrennte Liste von Bezeichnern, die als Werte von Attributen des DTD-Typs ID vorkommen.
(*ident*|*ident*|...)	Einer der aufgelisteten Bezeichner.

Eindeutige und voreingestellte Attributwerte

Mit Attributen der DTD-Typen ID, IDREF und IDREFS können Elemente in verschiedenen Teilen des Dokuments aufeinander Bezug nehmen. Die Werte von ID-Attributen müssen eindeutig sein, unabhängig vom Attributnamen.[22] Der Vorgabewert *default* regelt den Umgang mit fehlenden Attributen.

#REQUIRED	Das Attribut ist verpflichtend, es darf nicht fehlen.
#IMPLIED	Optionales Attribut ohne Defaultwert.
"*text*"	Optionales Attribut mit dem Defaultwert *text*.

Das folgende Beispiel zeigt eine DTD zum XML-Dokument time.xml (Listing 3.2). Diese DTD regelt, dass Stunden und Minuten genannt werden müssen, die Sekundenangabe aber optional ist. Auch die time-Attribute zone und summer sind optional. Wenn summer genannt wird, muss der Wert true sein.

Listing 3.8: DTD für XML-Dokumente mit Zeitangaben.

```
<!ELEMENT time (hour, minute, second?)>
<!ELEMENT hour (#PCDATA)>
<!ELEMENT minute (#PCDATA)>
<!ELEMENT second (#PCDATA)>
<!ATTLIST time
    zone    CDATA           #IMPLIED
    summer  (true)          "true"
>
```

Bezug zwischen XML-Dokument und DTD

Um ein XML-Dokument mit einer DTD zusammenzuführen, gibt es die folgenden Möglichkeiten.

- Das XML-Dokument nimmt selbst keinen Bezug auf die DTD, die in einer eigenständigen Textdatei gespeichert ist. Erst die Validierung verknüpft das XML-Dokument mit der DTD.
- Das XML-Dokument referenziert die DTD als externes Dokument wie im folgenden Beispiel time-external-dtd.xml. In der zweiten Zeile ist der Name

22 IDs werden beispielsweise zur Serialisierung mit JavaBeans (Seite 146) und XStream (Seite 154) gebraucht.

des Wurzelelements und die Textdatei angegeben, in der die DTD gespeichert ist.[23]

Listing 3.9: XML-Dokument mit Referenz auf eine externe DTD.

```
<?xml version="1.0"?>
<!DOCTYPE time SYSTEM "time.dtd">
<time zone="GMT+1" summer="true">
   <hour>12</hour>
   <minute>50</minute>
</time>
```

- Das XML-Dokument enthält eine interne DTD, wie im folgenden Beispiel `time-internal-dtd.xml`:

Listing 3.10: XML-Dokument mit einer internen (eingebetteten) DTD.

```
<?xml version="1.0"?>
<!DOCTYPE time [
    <!ELEMENT time (hour, minute, second?)>
    <!ELEMENT hour (#PCDATA)>
    <!ELEMENT minute (#PCDATA)>
    <!ELEMENT second (#PCDATA)>
    <!ATTLIST time
        zone    CDATA           #IMPLIED
        summer  (true)          "true"
    >
]>
<time zone="GMT+1" summer="true">
   <hour>12</hour>
   <minute>50</minute>
</time>
```

Validierung gegenüber einer DTD

Zur Validierung wird ein Werkzeug gebraucht, das XML-Dokumente und DTDs liest und abgleicht. Das frei verfügbare Kommandozeilenprogramm `xmllint` ist ein solches Werkzeug.[24]

Im folgenden Beispiel validiert `xmllint` das XML-Dokument `time-attribute.xml` (Listing 3.3) gegenüber der eigenständigen DTD-Datei `time.dtd` (Listing 3.8). Das Programm findet keine Fehler und gibt nichts aus, das Dokument ist also valide.[25]

```
$ xmllint --noout --dtdvalid time.dtd time-attribute.xml
$
```

23 Allgemein ist hier eine URL zulässig, die die DTD zur Verfügung stellt.

24 `xmllint` ist Teil des „XML C Parser Toolkit“, das ursprünglich für den Unix-Desktop Gnome entwickelt wurde. Die Bibliothek ist aber auch ohne Gnome verwendbar. Fertige, ausführbare Versionen von `xmllint` stehen für alle verbreiteten Systeme zur Verfügung.

25 Die meisten Programme aus dem Unix-Umfeld geben möglichst wenig aus. Keine Ausgabe bedeutet also, dass alles in Ordnung ist.

Das folgende Dokument enthält ein überzähliges Element `foobar`:

```
<?xml version="1.0"?>
<time zone="GMT+1" summer="true">
   <foobar>Junk</foobar>
   <hour>12</hour>
   <minute>50</minute>
</time>
```

`xmllint` entdeckt und meldet den Fehler. Das Dokument `time-foobar.xml` ist nicht valide bezüglich `time.dtd` (Listing 3.8):

```
$ xmllint --noout --dtdvalid time.dtd time-foobar.xml
time-foobar.xml:2: element time: validity error :
    Element time content does not follow the DTD,
    expecting (hour , minute , second?), got (foobar hour minute )
time-foobar.xml:3: element foobar: validity error :
    No declaration for element foobar
Document time-foobar.xml does not validate against time.dtd
$
```

Schwächen der DTD

Allerdings klassifiziert `xmllint` auch das folgende XML-Dokument als valide, obwohl es offensichtlich keine brauchbare Zeitangabe enthält:[26]

Listing 3.11: Gemäß DTD valides XML-Dokument mit einer unsinnigen Zeitangabe.

```
<?xml version="1.0"?>
<time zone="Abrakadabra Simsalabim" summer="true">
    <hour/>
    <minute>-6666666666</minute>
</time>
```

Hier zeigt sich eine der Schwächen von DTDs. Abgesehen von der eigenwilligen Syntax[27] ist die Ausdrucksmächtigkeit begrenzt. So kann der Inhalt von Textknoten und Attributwerten kaum kontrolliert werden.[28] Weiter gibt es keine Möglichkeit, ähnliche Teilstrukturen zusammenzufassen. Die Unterstützung von Namespaces fehlt, ebenso wie ein Mechanismus um unabhängige DTDs für Teildokumente zu definieren und wieder zu kombinieren.

XML-Schema

XML-Schreibweise für XML-Grammatiken

Eine XML-Schema-Definition (XSD, kurz „XML-Schema“) hat den gleichen Zweck wie eine DTD, erlaubt aber eine genauere Kontrolle. Abgesehen davon ist eine XSD selbst ein XML-Dokument. XSDs sind damit zwar für Menschen schwerer zu lesen als DTDs, aber maschinell mit den gleichen Mitteln zu verarbeiten wie alle XML-Dokumente. Zum Beispiel definiert die Grammatik `http://www.w3.org/2001/XMLSchema` den Aufbau von XML-Schema.[29]

26 Um Missverständnissen vorzubeugen: Das ist ein Problem der DTDs, nicht von `xmllint`.

27 Diese Syntax ist Teil des älteren Formalismus „SGML“, aus dem auch XML entstanden ist. SGML hat wenig Verbreitung gefunden.

28 Die einzige Ausnahme sind Attribute der DTD-Typen `ID`, `IDREF` und `IDREFS`, deren Attributwerte ausgewertet und abgeglichen werden.

29 Diese Grammatik kann sich selbst validieren.

Der Rahmen einer XSD sieht folgendermaßen aus:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    ...
</schema>
```

Nachbildung der Elementstruktur

Unter dem Wurzelelement findet sich eine Art Abbild der Struktur des beschriebenen XML-Dokuments. Elemente der Grammatik werden zu Elementen namens `element`. Das Dokument

```
<?xml version="1.0"?>
<time>
    <hour>12</hour>
    <minute>50</minute>
</time>
```

wird definiert durch die XSD

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="time">
        ...
            <element name="hour">...</element>
            <element name="minute">...</element>
        ...
    </element>
</schema>
```

Die Elemente des Dokuments (`time`, `hour`, `minute`) werden im Schema in der gleichen Anordnung zu `element`-Elementen mit entsprechenden `name`-Attributen.

Schema-Typen für den Inhalt von Elementen

So genannte Schema-Typen beschreiben den Inhalt von Elementen. Es gibt einfache und komplexe Schema-Typen (*simple types*, *complex types*).

Elemente, die nur Text enthalten, haben einen einfachen Schema-Typ. XML-Schema gibt eine ganze Reihe einfacher Schema-Typen vor, mit denen der Aufbau des Textinhalts eingeschränkt werden kann. Die einfachen Schema-Typen sind an Typen von Programmiersprachen angelehnt. Beispiele sind:

Einfache Schema-Typen

`string`	Beliebiger Text
`integer`	Ganze Zahl
`nonNegativeInteger`	Nicht negative ganze Zahl
`boolean`	`true` oder `false`
`double`	Gleitkommazahl

Die Stunden- und Minutenzahl in der Zeitangabe lässt sich damit genauer festlegen:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="time">
        ...
            <element name="hour" type="nonNegativeInteger">...</element>
            <element name="minute" ↘
                        ➞ type="nonNegativeInteger">...</element>
        ...
    </element>
</schema>
```

Einschränkung des Wertebereichs

Die einfachen Schema-Typen können noch weiter beschränkt werden durch zusätzliche Angabe von

- Grenzwerten (Attribute `minInclusive` und `maxExclusive`) bei Zahlen,
- regulären Ausdrücken (Attribut `pattern`) bei Strings und
- Aufzählungen (Attribut `enumeration`) bei Strings.

Im folgenden Beispiel werden die Stunden- und die Minutenangabe auf sinnvolle Werte (0–23 und 0–59) begrenzt:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="time">
        ...
            <element name="hour">
                <simpleType>
                    <restriction base="nonNegativeInteger">
                        <maxExclusive value="24"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="minute">
                <simpleType>
                    <restriction base="nonNegativeInteger">
                        <maxExclusive value="60"/>
                    </restriction>
                </simpleType>
            </element>
        ...
    </element>
</schema>
```

Komplexe Schema-Typen für zusammengesetzte Strukturen

Elemente mit einem anderen Inhalt als Textknoten haben einen komplexen Schema-Typ. Das betrifft alle Elemente mit geschachtelten Elementen als Kindern. „Ordnungsindikatoren“ legen die Anordnung der Kindelemente fest:

`all`	Jedes der aufgezählten Elemente einmal in beliebiger Reihenfolge.
`sequence`	Jedes der aufgezählten Elemente in der angegebenen Reihenfolge.
`choice`	Eines der aufgezählten Elemente.

Damit kann das XML-Schema für Zeitangaben weiter vervollständigt werden.

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="time">
        <complexType>
            <sequence>
                <element name="hour">...</element>
                <element name="minute">...</element>
            </sequence>
        </complexType>
    </element>
</schema>
```

Anzahl Vorkommen von Kindelementen

Die Anzahl der Vorkommen kann mit den Attributen `minOccurs` und `maxOccurs` eingegrenzt werden, deren Werte vorzeichenlose ganze Zahlen sind.[30] Für `maxOccurs` ist außerdem die Angabe `unbounded` erlaubt, wenn die Anzahl Vorkommen beliebig ist. Die Voreinstellung bei fehlender Angabe ist 1.

Definition von Attributen

XSD-Elemente `attribute` definieren Attribute des Dokuments. Auch `attribute`-Elemente gehören zu Schema-Typen. Das folgenden Beispiel ergänzt den Schema-Typ von `time`-Elementen um die beiden Attribute `zone` und `summer`:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="time">
        <complexType>
            <sequence>...</sequence>
            <attribute name="zone">...</attribute>
            <attribute name="summer">...</attribute>
        </complexType>
    </element>
</schema>
```

Für Attributwerte stehen die gleichen einfachen Schema-Typen (*simple type*) zur Verfügung wie für Textknoten.[31]

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="time">
        <complexType>
            <sequence>...</sequence>
            <attribute name="zone">
                <simpleType>
                    <restriction base="string">
                        <pattern value="[A-Z]{3}[-+][0-9]{1,2}"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="summer" type="boolean" default="true"/>
        </complexType>
    </element>
</schema>
```

30 Beim Ordnungsindikator `all` ist `maxOccurs` auf den Wert 1 fixiert. Für `minOccurs` ist nur 0 oder 1 zulässig.

31 Der reguläre Ausdruck für die Werte von `zone` definiert die textuelle Form von Zeitzonen, lässt aber immer noch sinnlose Angaben zu, wie zum Beispiel `XXX-99`. Dieser reguläre Ausdruck ist ein Kompromiss zwischen Aufwand und Korrektheit.

Schema-Typen können mit Typnamen versehen und dann mehrfach referenziert und geschachtelt werden. Im folgenden Beispiel werden Zeitangaben um ein optionales Sekunden-Element ergänzt, dessen Inhalt den gleichen Typ wie das Minuten-Element hat. Um eine Wiederholung zu vermeiden, ist dieser Typ explizit als `Number60` definiert.

Daraus ergibt sich ein technisches Problem: Im Wurzelelement wurde bisher die Schema-Grammatik als Default-Namespace definiert (siehe Seite 174). Der selbst definierte Typ `Number60` existiert dort natürlich nicht. Ein Namespace-Präfix für XML-Schema und eine eindeutige Zuordnung der Element- und Attributnamen lösen das Problem:

Listing 3.12: `time.xsd`, XML-Schema für XML-Dokumente mit Zeitangaben.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="time" type="Time"/>
    <xsd:complexType name="Time">
        <xsd:sequence>
            <xsd:element name="hour">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:nonNegativeInteger">
                        <xsd:maxExclusive value="24"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="minute" type="Number60"/>
            <xsd:element name="second" type="Number60" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="zone">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:pattern value="[A-Z]{3}[-+][0-9]{1,2}"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="summer" type="xsd:boolean" default="true"/>
    </xsd:complexType>
    <xsd:simpleType name="Number60">
        <xsd:restriction base="xsd:nonNegativeInteger">
            <xsd:maxExclusive value="60"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Validierung gegenüber einem XML-Schema

Zur Validierung stehen viele Werkzeuge zur Verfügung. Das Kommandozeilenwerkzeug `xmllint` kann auch mit XSD umgehen, um beispielsweise `time-attribute.xml` (Listing 3.3) zu validieren:

```
$ xmllint --noout --schema time.xsd time-attribute.xml
time-attribute.xml validates
$
```

`xmllint` erkennt jetzt das XML-Dokument `time-funny.xml` (Listing 3.11) mit unsinnigem Inhalt, das mit einer DTD fälschlich als valide klassifiziert wurde, als unzulässig:

```
$ xmllint --noout --schema time.xsd time-funny.xml
time-funny.xml:2: element time: Schemas validity error :
    Element 'time', attribute 'zone': [facet 'pattern']
    The value 'Abrakadabra Simsalabim' is not accepted by the pattern ↘
                          ➞ '[A-Z]{3}[-+][0-9]{1,2}'.
time-funny.xml:2: element time: Schemas validity error :
    Element 'time', attribute 'zone':
    'Abrakadabra Simsalabim' is not a valid value of the local atomic ↘
                          ➞ type.
time-funny.xml:3: element hour: Schemas validity error :
    Element 'hour': '' is not a valid value of the local atomic type.
time-funny.xml:4: element minute: Schemas validity error :
    Element 'minute': '-6666666666' is not a valid value of the ↘
                          ➞ atomic type 'Number60'.
time-funny.xml fails to validate
$
```

Bezug zwischen XML-Dokument und -Schema

Wie bei DTDs gibt es verschiedene Möglichkeiten, um ein XML-Dokument mit einem XML-Schema in Verbindung zu bringen (siehe Seite 177):

- Das XML-Dokument weist keinen Bezug zu einem Schema aus. Das Schema wird erst bei der Validierung festgelegt. Nach diesem Verfahren gehen die bisher gezeigten Beispiele vor.
- Das XML-Dokument referenziert selbst ein bestimmtes Schema. Ein Paar von Attributen im Wurzelelement liefert die nötige Information:

  `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
  : Definiert das Präfix `xsi` für die nachfolgende Angabe eines XML-Schemas.

  `xsi:noNamespaceSchemaLocation="`*filename*`"`
  : Legt fest, dass die Datei *filename*[32] das XML-Schema für alle Elemente und Attribute ohne Namespace-Präfix beisteuert.

  Das folgende XML-Dokument bezieht sich auf die Schema-Datei `time.xsd` (Listing 3.12):

Listing 3.13: XML-Dokument mit Referenz auf ein XML-Schema.

```
<?xml version="1.0"?>
<time
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="time.xsd"
    zone="GMT+1"
    summer="true">
    <hour>12</hour>
    <minute>50</minute>
</time>
```

32 Der Attributwert kann eine beliebige URL nennen, unter der das Schema zu finden ist. Eine lokale Datei ist nur ein einfacher Sonderfall.

Anders als bei DTDs wäre ein Dokument mit einem eingebetteten Schema nicht sehr sinnvoll. Das Schema müsste sich in diesem Fall als Teil des Dokuments selbst validieren.[33]

Java bietet selbst die Mittel zur Validierung, wie der nächste Abschnitt zeigt. Ein Java-Programm kann damit die Aufgabe von Werkzeugen wie `xmllint` übernehmen.

3.2 Arbeit mit dem DOM

Objektrepräsentation einer XML-Struktur

Ein „Document Object Model“ (DOM) ist eine programminterne Darstellung eines kompletten XML-Dokuments. Die verschiedenen Bestandteile des XML-Dokuments werden im DOM durch Java-Objekte repräsentiert. Die Objekte sind als Baum mit der gleichen Struktur wie das Dokument organisiert.[34]

Ein DOM-Parser liest ein XML-Dokument und liefert die Baumstruktur. Der Baum kann dann durchsucht, verändert und schließlich wieder als XML-Dokument oder in einer anderen Form ausgegeben werden.

Merkmale eines DOM-Parsers

Diese Vorgehensweise holt das *komplette* Dokument in das Programm. Das hat Vor- und Nachteile. Der wesentliche Vorteil ist die Möglichkeit, beliebige Stellen im Baum gleichzeitig anzusprechen. Ein Nachteil ist die Begrenzung des Datenvolumens, das komplett in den Speicher passen muss. Eine Alternative mit entgegengesetzten Eigenschaften ist die sequenzielle Verarbeitung mit einem SAX-Parser (Seite 201).

3.2.1 DOM-Parser

DOM-Parser in der Java-Bibliothek

Einen XML-Parser zu schreiben ist keine einfache Aufgabe. Glücklicherweise gibt es eine ganze Reihe fertiger XML-Parser. Auch die Java-Bibliothek enthält einen „DOM-Parser“.[35]

DOM-Parser sind Klassen, die von der abstrakten Basisklasse `DocumentBuilder` im Package `javax.xml.parsers` abgeleitet sind. `DocumentBuilder` werden allerdings nicht mit Konstruktoren erzeugt, sondern mit der Factory-Methode `newDocumentBuilder` der Klasse `DocumentBuilderFactory`. Zunächstmuss also eine `DocumentBuilderFactory` beschafft werden:

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

Der zweite Schritt erzeugt mithilfe der Factory einen neuen DOM-Parser:

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

33 DTDs mit ihrer eigenen Syntax mischen sich nicht mit den übrigen Elementen. Das Problem stellt sich dort daher nicht.

34 Streng genommen gilt das nicht für Attribute eines Elements, die keiner bestimmten Anordnung unterliegen.

35 Der DOM-Parser in der Java-Bibliothek beruht auf dem Open-Source-Parser „Xerces“ der Apache Foundation.

Ein DOM-Parser definiert als wichtigste Methode `parse`, die ein XML-Dokument liest und die Wurzel des DOM liefert. Auf den Typ des Wurzelknotens, `Document`, geht der nächste Abschnitt ein.

```
Document document = builder.parse(...);
```

Aufruf eines DOM-Parsers

Das folgende Beispielprogramm zeigt den Ablauf:

Listing 3.14: Lesen eines XML-Dokumentes in ein DOM.

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

public class ParseDOM {
    public static void main(String... args) throws ↘
                        ➙ ParserConfigurationException, ↘
                        ➙ SAXException, IOException {
        DocumentBuilderFactory factory = ↘
                        ➙ DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(args[0]);
    }
}
```

Es gibt auch andere Wege eine XML-Datei in ein DOM zu parsen. Das hier gezeigte Verfahren hat den Vorteil, dass es sich mit wenig Aufwand auf sequenzielle Verarbeitung (Seite 201) übertragen lässt.[36]

3.2.2 Validierung

Einbindung einer DTD zur Validierung

Der Standard-DOM-Parser von Java setzt ein wohlgeformtes XML-Dokument voraus, validiert das Dokument aber in der Voreinstellung nicht.

DTD

Validierung gegenüber einer DTD lässt sich in der `DocumentBuilderFactory` mit dem Setter `setValidating` anfordern. Die Factory produziert im folgenden Beispiel einen validierenden Parser:

Listing 3.15: Validieren eines XML-Dokumentes gegenüber einer externen oder internen DTD.

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

36 Zum Beispiel spezifiziert DOM-Version 3 das Interface „Load and Save“, dessen Java-Implementierung im Package `http://org.w3c.dom.ls` zu finden ist. „Load and Save“ ist allerdings DOM-spezifisch.

Copyright