XSD - XML Schema Definition

Definieren von XML-Dokumenten

Michael Dienert

15. September 2016

Inhaltsverzeichnis

1 Was sind XSD Dateien und warum soll man das lernen?

1.1 XSD Dateien und Anwendungen

- XSD ist die Abkürzung von *XML Schema Definition*.
- XSD ist selbst wieder eine XML-Anwendung.
- D.h. XSD-Dokumente müssen immer *wohlgeformte* XML-Dokumente sein.
- XSD beschreibt die **Struktur** eines XML-Dokuments.
- Mit XSD kann man die Struktur **genauer** beschreiben als mit der DTD.

Einige Erweiterungen gegenüber DTD:

- Genaue Angaben **wie oft** ein Element vorkommen darf.
- Datentypen
- XSD erlaubt auch eigene, komplexe Datentypen zu definieren → Objekte.
- Datentypen können von bestehenden Datentypen abgeleitet werden → Vererbung.
- XSD ist **der** weltweite Standard, mit dem XML-Dateien definiert werden.
- Java-XML-Bindings erzeugen aus einer XSD-Datei eine Klasse, deren Objekte exakt die zugehörige XML-Datei abbilden.

- Man kann die Java-Objekte mit allen Attributen in die XML-Datei sichern (nur die Attribute!).
- Man kann aus der XML-Datei wieder die Java-Objekte erzeugen.
- Man kann sogar aus einer Java-Klasse eine XSD-Datei erzeugen!

1.1.1 Ein XSD - Beispiel

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="kontakte">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Person" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="adresse" type="xs:string" minOccurs="0" />
     </xs:sequence>
    <xs:attribute name="id" type="xs:long" use="required" />
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

- **`xs:element`** Element mit Daten
- Attribute von `xs:element`:

  name Elementname

  type Datentyp: siehe nächste Folie

  minOccurs Mindestanzahl; `minOccurs` weglassen: Anzahl=1

  maxOccurs Maximalanzahl; `maxOccurs` weglassen: Anzahl=1; `maxOccurs=unbounded` = beliebig viele

XSD-Standard-Datentypen - Simple Types

xs:string

xs:decimal

xs:integer

xs:float

xs:boolean

xs:date

xs:time

xs:anyType

```xml
<xs:element name="name" type="xs:string" />

Beispiel:
<name>alfred</name>

_____________________________________________________
<xs:element name="adresse" type="xs:string" />

Beispiel:
<adresse>friedrichstrasse 51, 79098 freiburg</adresse>
```

```
<xs:element  name="id" type="xs:long" use="required" />

Beispiel:
<id>1234</id>
```

- Elemente, die *Kindselemente* oder *Attribute* oder beides enthalten werden mit `xs:complexType` definiert
- Ist die Reihenfolge der Kinder fest, werden diese zusätzlich in ein `xs:sequence` - Element eingebettet.
- Ist die Reihenfolge beliebig, nimmt man stattdessen das Element `xs:choice`
- Selbstverständlich darf man auch innerhalb eines `xs:sequence`-Elements `xs:choice`-Elemente einbetten und umgekehrt.
- man kann so feste und beliebige Reihenfolge mischen.

```
<xs:element name="Person" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string" />
   <xs:element name="adresse" type="xs:string" minOccurs="0" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
<Person>
    <name>neuman, alfred e.</name>
    <adresse>locaRoca 666, 1234 dancelingen</adresse>
</Person>
```

- Das Element `xs:attribute` legt die *Attribute* eines Elements fest.
- Es kann nur innerhalb einer `xs:complexType` - Umgebung verwendet werden.
- `xs:attribute` hat folgende Attribute:

  use kann den Wert `"required"` oder `"optional"` haben.

  type legt den Datentyp fest. Mögliche Basistypen sind die selben wie bei den Elementen.

- Enthält `xs:complexType` ein `xs:sequence` - Element, **muss** das Element `xs:attribute` nach dem `xs:sequence` - Element stehen!

Element mit Attribut ohne Kindselemente und ohne Textnoten

```
<xs:element name="timestamp">
 <xs:complexType>
  <xs:attribute name="utc" type="xs:long"/>
 </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
    <timestamp utc="123456789"/>
```

Element mit Attribut und Kindselementen

```
<xs:element name="Person" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string" />
   <xs:element name="adresse" type="xs:string" minOccurs="0" />
  </xs:sequence>
  <xs:attribute  name="id" type="xs:long" use="required" />
 </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
    <Person id="19700101">
        <name>neuman, alfred e.</name>
        <adresse>locaRoca 666, 1234 dancelingen</adresse>
    </Person>
```

Element mit Attribut und Textknoten

```
<xs:element name="termin">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute name="utc" type="xs:long"/>
   </xs:extension>
  </xs:simpleContent>
  <xs:attribute name="utc" type="xs:long"/>
 </xs:complexType>
</xs:element>
```

Beispiel eines validen Elements:

```
    <termin utc="123456789">klassenarbeit</termin>
```

- Die XSD-Datei beschreibt eine xml-Datei, die beliebig viele (`maxOccurs="unbounded"`) <Person>-Elemente enthalten darf.
- Jedes <Person>-Element muss ein Atrribut **id** haben.
- Jedes <Person>-Element enthält **genau ein** (weder minOccurs noch maxOccurs vorhanden) Element <name>.
- Jedes <Person>-Element enthält **optional** (`minOccurs=0`, maxOccurs nicht vorhanden, d.h `maxOccurs="1"`) ein Element <adresse>.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<kontakte>
    <Person id="19631223">
        <name>dienert, michael</name>
        <adresse>waelderstrasse 7, 79341 kenzingen</adresse>
    </Person>
    <Person id="19700101">
        <name>neuman, alfred e.</name>
        <adresse>locaRoca 666, 1234 dancelingen</adresse>
```

```
    </Person>
    <Person id="20000210">
        <name>bosak, jon</name>
        <adresse>
           4150 Network Cir, Santa Clara, CA USA
        </adresse>
    </Person>
</kontakte>
```

Simple Types anpassen

- Simple Types können durch Einschränkungen *angepasst* werden.
- Die Definitionen dieser Einschränkungen werden *Facets* genannt.

Aufzählungen - Enumerations

```
<xs:simpleType name="wochentag">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Montag" />
  <xs:enumeration value="Dienstag" />
  <xs:enumeration value="Mittwoch" />
  ...
 </xs:restriction>
</xs:simpleType>
```

Anzahl Zeichen in einem String

```
<xs:simpleType name="password">
 <xs:restriction base="xs:string">
  <xs:minLength value="6" />
  <xs:maxLength value="8" />
 </xs:restriction>
</xs:simpleType>
```

Ein Facet für E-Mail-Adressen

```
<xsd:simpleType name="emailAddress">
 <xsd:restriction base="xsd:string">
   <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"/>
 </xsd:restriction>
</xsd:simpleType>
```

Java-Code generieren mit dem Kommando `xjc`:

```
xjc kontakte.xsd
```

Direktes Erzeugen der Quellen innerhalb eines NB-Projekts:

File-Menue → New File → Categorie: XML → Type: JAXB Binding

- JAXB erzeugt eine Klasse, mit dem Namen des Wurzelelements: `Kontakte.java`
- Die vielen <Personen>-Elemente bildet JAXB auf eine `ArrayList` ab, die Objekte vom Typ `Kontakte.Person` enthalten darf. d.h. `Person` ist innere Klasse von `Kontakte`.
- Die innere Klasse **Person** wiederum ist nichts anderes wie eine **JavaBean** mit den Eigenschaften:

- `String name;`
- `String adresse;`
- `long id;`

- Marshalling: ähnlich Serialisierung von Objekten; Orignalbedeutung: Einweisen eines Flugzeugs auf dem Flugfeld zur Startbahn durch den Marshall.
- Code-Schnipsel:

```
// create JAXB context and instantiate marshaller
JAXBContext context = JAXBContext.newInstance(Kontakte.class);
Marshaller m = context.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

// Write to System.out
m.marshal(kontakte, System.out);

// Write to File
m.marshal(kontakte, new File("/tmp/kontakte.xml"));
```

- Unmarshalling: Erzeugung von Objekten der mit `xjc` generierten Klassen. Hier:`Kontakte` und `Kontakte.Person`
- Code Schnipsel:

```
private List<Kontakte.Person> personListe;

JAXBContext context = JAXBContext.newInstance(Kontakte.class);
Unmarshaller u = context.createUnmarshaller();

Kontakte kontakteNeu = (Kontakte) u.unmarshal(
   new File("/tmp/andereKontakte.xml"));

 personListe = kontakteNeu.getPerson();
```