XML – Merkblatt 4 – JAXB

Einführung Die Java Architecture for XML Binding (JAXB) ist ein Java-Standard, um XML-Dokumente mit Java verarbeiten zu können ohne tiefe Kenntnisse über XML besitzen zu müssen. Es wird lediglich ein XML-Schema der XML-Dokumente benötigt. Zuerst werden aus dem XML-Schema Java-Klassen erzeugt, die das Schema repräsentieren (binding the schema). Dann wird mit diesen Java-Klassen ein Java-Programm implementiert, mit dem die Objekte dieser Klassen aus gültigen XML-Dokumenten eingelesen (unmarshal) oder im Programm erzeugte Objekte, in XML-Dokumente geschrieben werden (marshal). Der JAXB-Standard ist Teil des JDK seit Version 6 Update 4.

Schema an Klassen binden Die Java-Klassen werden mit einem Compiler aus dem XML-Schema erzeugt: `xjc`. Da diese Klassen automatisch generiert werden, sollten man sie nicht manuell ändern. Am besten ordnet man sie mit Option `-p`) einem eigenen Paket zu. Im folgenden nutzen wir das `sports-event`-XML-Schema aus dem letzten Merkblatt. Lediglich `xmlns="mein.namensraum"` im Wurzelelement ist hinzugefügt, um einen Fehler zu vermeiden.

```
ape@melkor:sport-java/ >ls
sport-event.xsd
pape@melkor:sport-java/ >xjc -p sport.jaxb sport-event.xsd
Ein Schema wird geparst ...
Ein Schema wird kompiliert ...
sport/jaxb/EventType.java
sport/jaxb/ObjectFactory.java
sport/jaxb/SportEvents.java
sport/jaxb/package-info.java
pape@melkor:sport-java/ >ls
sport  sport-event.xsd
```

Welche Klassen erzeugt werden, hängt vom verwendeten Compiler ab. Für jeden komplexen Typ gibt es normalerweise eine Klasse (value class). Diese enthält nur Werte, wie sie im Schema beschrieben sind. Teilweise werden innere Klasse bei anonymen Typen erzeugt. Die Objekte für komplexen Typen müssen indirekt mit den Methoden der `ObjectFactory.java` statt direkt mit einem Konstruktor erzeugt werden. Mit getter- und setter-Methoden, werden bei den Objekten die Werte für Elemente oder Attribute gesetzt oder ausgelesen. Die Namen dieser Methoden orientieren sich an den Namen im Schema. Bei mehrfachen Vorkommen wird ein List-Objekt zum Speichern der Werte verwendet.

Der JAXB-Standard definiert auch die Abbildung zwischen Java-Datentypen und den atomaren Typen des XML-Schemas.

Atomic type	Javatyp
string, anyURI, normalizedString	java.lang.String
int, unsignedInt	int
float	float
double	double
decimal	java.math.BigDecimal
dateTime, time, date	javax.xml.datatype.XMLGregorianCalendar
boolean	boolean

Bei primitiven Datentypen kann auch zum Beispiel für `int` die Wrapper-Klasse `Integer` auftreten. Insbesondere, wenn das Element oder Attribut optional ist.

Für den anonymen komplexen Typ von `sport-events` wurde im wesentlichen folgende Klasse erzeugt. Deren Verwendung ist einfach und setzt keine XML-Kenntnisse voraus.

```
public class SportEvents {
    protected String city;
    protected List<EventType> event;
    protected BigDecimal version;
    public String getCity() {..}
    public void setCity(String value) {..}
    public List<EventType> getEvent() {..}
    public BigDecimal getVersion() {..}
    public void setVersion(BigDecimal value) {..}
}
```

XML-Dokument einlesen Um ein XML-Dokument einzulesen und mit Java zu verarbeiten können, müssen die Textinhalte in Java-Objekte umgewandelt werden (unmarshalling). Dazu sind drei Schritte nötig:

1. Ein `JAXBContext`-Objekt als Ankerpunkt für die JAXB-API erzeugen. Das Java-Paket der vom Compiler erzeugten Klassen muss dabei angegeben werden.

2. Ein `Unmarshaller`-Objekt erzeugen, mit dem ein XML-Dokument in Java-Objekte umgewandelt werden kann.

3. Der Aufruf von `unmarshall(Reader r)`, um ein über ein Reader-Objekt das XML-Dokument einzulesen und in Objekte umzuwandeln. Es wird ein `JAXBElement` zurückgegeben.

4. Das `JAXBElement` kann nicht immer auf das Werteobjekt gecastet werden. Deswegen muss das Werteobjekt erst noch mit `JAXBIntrospector.getValue(JAXBElement e)` geholt werden.

Dazu sind in unserem Beispiel drei Code-Zeilen nötig:

```
JAXBContext context = JAXBContext.newInstance("sports.jaxb");
Unmarshaller unmarshaller = context.createUnmarshaller();
SportEvents events = (SportEvents) JAXBIntrospector.getValue(
                unmarshaller.unmarshal(new File("sport-events.xml")));
```

Die benötigten Klassen befinden sich im Paket `javax.xml.bind`.

Über das `events`-Objekt und mit den Zugriffsmethoden, kann auf die Werte des XML-Dokuments zugegriffen werden. Sie werden intern baumartig als Java-Objekte gespeichert. Folgendes Programmfragment gibt für eine Stadt die Namen aller Sportereignisse aus.

```
System.out.println("Sportereignisse in " + events.getCity());
for (EventType event : events.getEvent()) {
  System.out.println( event.getName() );
}
```

XML-Dokument schreiben Um ein XML-Dokument aus Java zu erzeugen, muss ein Programm implementiert werden, welches alle XML-Daten ausgehend vom Wurzelelement baumartig mit den zugehörigen Java-Objekte erzeugt. Die Objekte müssen mit einer Instanz von `ObjectFactory` erzeugt werden. Für einige spezielle Datentypen wie `dateTime` müssen die zugehörigen Java-Objekte über eine `DatatypeFactory`erzeugt werden. Sie ist Teil der Java-XML-Standard-APIs und befindet sich in `java.xml.datafactory`.

```
ObjectFactory factory = new ObjectFactory();
DatatypeFactory datatypeFactory = DatatypeFactory.newInstance();
SportEvents events = factory.createSportEvents();
events.setCity("Berlin");
events.setVersion(new BigDecimal(1.5)) ;
EventType event = factory.createEventType();
event.setName("Marathon");
event.setStart(datatypeFactory.newXMLGregorianCalendar("2017-09-24"));
events.getEvent().add(event);
```

Anschließend kann analog zum Unmarshalling-Prozess der Objektbaum ausgehend vom Objekt des Wurzelelements als XML serialisiert in einen Reader geschrieben werden.

```
JAXBContext context = JAXBContext.newInstance("sport.jaxb");
Marshaller marshaller = context.createMarshaller();
marshaller.marshal(events, new File("sport-events-from-java.xml"));
```

Das erzeugte XML-Dokument sieht – zusätzlich leserlich formatiert – wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:sport-events xmlns:ns2="mein.namensraum" version="1.5">
  <city>Berlin</city>
  <event>
    <name>Marathon</name>
    <start>2017-09-24</start>
  </event>
</ns2:sport-events>
```

Ergänzende Literatur Im [1] gibt es eine Einführung zum JAXB-Standard. Mehr Details finden sich in der aktuellen Spezifikation [2].

Literatur

[1] Java Architecture for XML Binding. Technical report, Oracle. https://docs.oracle.com/javase/tutorial/jaxb/index.html.

[2] The Java™ Architecture for XML Binding (JAXB) 2.2. Technical report, December 2009. http://download.oracle.com/otndocs/jcp/jaxb-2.2-mrel2a-oth-JSpec/.