Java spricht

Eine Lanze für XML brechen

Phillip Ghadir

Heute – zehn Jahre nach der ersten Vorstellung von JAXB im JavaSPEKTRUM – möchte ich noch einmal eine Lanze für XML brechen. Auch wenn JSON, der sympathische Cousin von XML, immer mehr Freunde gewinnt, bietet XML immer noch einiges, wenn auch selten genutztes Potenzial. In dieser Kolumne möchte ich ein paar Aspekte von XML noch mal in Erinnerung rufen.

In der Ausgabe 05/2002 wurde erstmals JAXB im JavaSPEKTRUM [Krüg02] vorgestellt. In 2004 war XML eines der Hauptthemen. Heute – zehn Jahre später – begegnen uns immer noch Projekte, in denen die Objekt-Serialisierung hakt. Ob aufgrund der Größe der serialisierten Daten, des Hauptspeicherverbrauchs während des Bindens oder aufgrund der Rechenzeit für Transformationen – es gibt genügend Gründe, sich das Thema noch mal anzuschauen. Hier insbesondere aufgrund der verfügbaren Werkzeugkette außerhalb der Java-Welt.

Domänenklassen – immer notwendig?

Kennen Sie mehrschichtige Anwendungen, die ihren Input in eine interne Objekt-Repräsentation umwandeln, um ihn dann durch die verschiedenen Schichten zu reichen? Wie sieht eine gute Objekt-Repräsentation aus, die keinerlei eigene Funktionalität beinhaltet? Welche Daseinsberechtigung hat eine Objekt-Repräsentation ohne eigene objektbezogene Fachlogik?

Wenn Sie auf die letzten beiden rhetorischen Fragen keine guten Antworten finden, könnten Sie prüfen, wie sich Ihre Architektur verbessern würde, wenn Sie auf die De-/Serialisierung von XML-Daten verzichten.

Wer XML-Daten liest, sie als Java-Objekte repräsentiert und dann wieder zu praktisch gleichem XML exportiert, ohne programmatisch Daten zu manipulieren, kann tendenziell auf die verschiedenen Kopier- und (De-)Serialisierungsvorgänge verzichten, wenn die Objekte keine Fachlogik implementieren.

Sollten Anpassungen von Input- und Output-XML nötig werden, bietet das Java SDK seit Langem für den Fall der Fälle mit `javax.xml.transform.*` die Infrastruktur, um strukturelle Abbildungen zu definieren. Wir haben in der Vergangenheit gute Erfahrungen damit gemacht, einfache Anpassungen direkt auf der XML-Darstellung vorzunehmen.

```
public static Transformer getTransformer(InputStream xslt )
    throws TransformerConfigurationException {
  TransformerFactory factory = TransformerFactory.newInstance();
  if ( xslt != null ) {
    return factory.newTransformer(new StreamSource(xslt));
  } else {
    return factory.newTransformer();
  }
}
public static void transform(
    InputStream xmlInput,
    InputStream xslt,
    OutputStream out) throws Exception {
  getTransformer( xslt ).transform(
    new StreamSource( xmlInput ), new StreamResult( out ));
}
```

Listing 1: Mit transform() kann jeder XML-Input einfach mit XSLT transformiert werden

Listing 1 zeigt, wie mit einfachen Java SDK-Mitteln XML transformiert werden kann. So kommt man auch ohne den Umweg über ein explizites XML-Binding recht weit.

Auch der Direkt-Zugriff auf einzelne Werte im XML kann sehr einfach per XPath erfolgen. Mit Hilfe der in Listing 3 abgebildeten Methode `getStringValuesForXPath()` sieht das Iterieren über eine Menge von Werten aus dem XML-Dokument sehr einfach aus (s. Listing 2).

```
for ( String s : getStringValuesForXPath(
    "//vertrag/vertragsnummer", xmlInputStream ) ) {
  System.out.println( s ); // beliebige Verarbeitung
}
```

Listing 2: Iterieren über per XPath ermittelte Werte aus einem XML-Dokument

Wobei die Methode `getStringValuesForXPath()` wiederum an zwei weitere Methoden delegiert. Die Methode `getValuesForXPath()` kapselt einfach den Aufruf der XPath-Funktionalität aus dem Package `javax.xml.xpath` und liefert ein `NoteListIterable` zurück, das dann mit Hilfe der Konverter und per *foreach* wie in Listing 2 dargestellt verwendet werden kann. Die zugrunde liegende Struktur verdeutlicht Abbildung 1.

```
private static Iterable<String> getStringValuesForXPath(
    String xPath, InputStream inputStream)
    throws XPathExpressionException {
  return convertNodes2Strings(
    getValuesForXPath( xPath, inputStream ) );
}
```

Listing 3: Eine Implementierung für das Extrahieren des Textinhalts aus einem DOM-Knoten besteht aus simplen Methoden

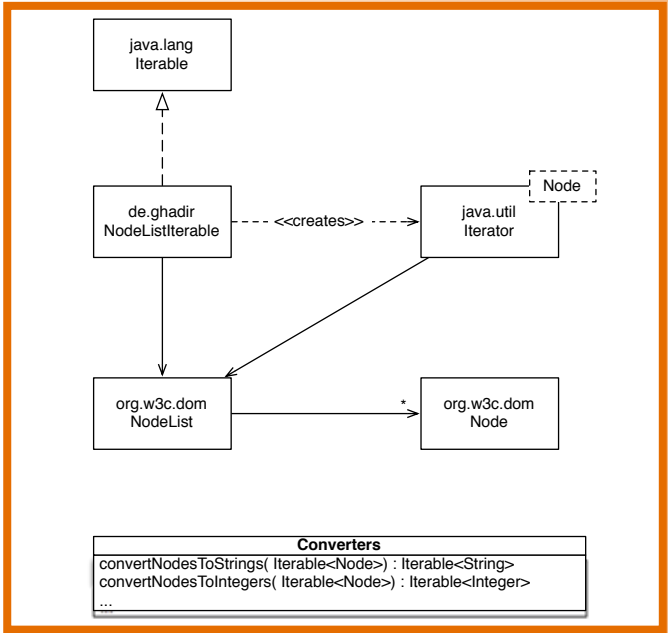

Abb. 1: Das Klassendiagramm zeigt den Zusammenhang zwischen selbstgeschriebener Funktionalität und im Java SDK enthaltenen XML-Klassen

Mit einer solchen Infrastruktur ist es möglich, Fachlogik im Backend zu realisieren, ohne das Domänenmodell in Form von Java-Klassen auszuprägen. Statt dessen kann die Fachlogik direkt auf die Daten in der gewünschten XML-Struktur zugreifen.

Anticorruption-Layer auch ohne Domänenklassen

Bleibt die Frage, wie die Entkopplung von anderen Domänenmodellen erfolgen kann, wenn wir keine Objekt-Repräsentationen realisieren. Dafür bieten sich XSLT-Transformationen an, die an den Integrationsschichten externe auf interne Strukturen (und umgekehrt) abbilden.

Durch den Aufruf der in Listing 1 dargestellten `transform()`-Methode erhält man mit der passenden Transformation eine sehr schlanke Abbildung zwischen unterschiedlichen Domänenmodellen. Beispielsweise hilft die Namenskonvention `<ext>-2-<int>.xslt` für die Inputs und `<int>-2-<ext>.xslt` für die Outputs, wobei `<int>` und `<ext>` für die Domänenmodelle stehen.

Warum XML noch nicht out ist

Auch wenn heute häufig gern JSON eingesetzt wird, bietet XML manche Eigenschaft, die man z. B. bei JSON schmerzlich vermisst. Zu allererst ist XML immer noch ein selbstbeschreibendes Format, zur menschenlesbaren Aufbereitung von beliebigen Strukturen. Das Schlüsselwort ist selbstbeschreibend und meint damit nicht XML-Tags um jedes einzelne Datum.

Für XML ist wohldefiniert,

- wie das Encoding angegeben wird.
- wie der Dokumenttyp definiert wird.
- welchem Schema ein Dokument entspricht.
- wie Links zu anderen Ressourcen anzugeben sind.
- wie zusätzliche Verarbeitungsanweisungen eingefügt werden können.
- wie Kommentare aussehen.
- wie Namensräume angegeben werden können.
- dass jeder Knoten Kind-Elemente und Attribute aus unterschiedlichen Namensräumen enthalten kann.
- wie mittels XPath Selektionen von Dokumentteilen formuliert werden.
- wie mit XSLT XML transformiert werden kann.
- wie man einem Element eine ID gibt oder auf diese verweist.

Zugegeben die Familie von Standards, Empfehlungen und Spezifikationen rund um XML ist komplex, umfangreich und nicht immer leichtgewichtig, aber dennoch ist XML weit verbreitet. Es existiert eine gute XML-Verarbeitungsinfrastruktur auf praktisch jedem Client-Arbeitsplatz, der über einen gängigen Webbrowser verfügt.

Darüber hinaus ist die Verarbeitung von XML in den gängigen Programmiersprachen bestens unterstützt. Das Java SDK bringt alles mit, was man sich für die XML-Verarbeitung benötigt.

Mehr Akzeptanz für unsere Dokumente erreichen

In vielen Java-Projekten wird gern ein Faktor vernachlässigt, der die allgemeine Akzeptanz unserer XML-Dokumente bei vielen Interessenvertretern schmälert. Nicht jeder im Fachbereich ist bereit, zwischen den vielen spitzen Klammern nach Nutzinhalt zu suchen.

Dabei könnte das Betrachten im Webbrowser unseres Vertrauens (Firefox, Safari oder selbst der Internet-Explorer) sehr angenehm sein, wenn wir die Darstellung mittels CSS aufpeppen würden und gegebenenfalls mit XSLT so ergänzten, dass die Darstellung mit CSS definiert werden kann.

Technisch ist die Infrastruktur komplett, sobald wir für die Darstellung benötigte Ressourcen über das Netz erreichbar ablegen. Dann genügt es, ins XML-Dokument Processing-Instructions mit der entsprechenden Stylesheet-Angabe einzufügen. Dies erfolgt über

```
<?xml-stylesheet type="text/xml" href="___SOME_XSLT_URL____ " ?>
<?xml-stylesheet type="text/css" href="___SOME_CSS_URL____" ?>
```

Abb. 2: Bildschirmfoto des XML-Dokuments im Browser – ohne Stylesheet-Anweisungen

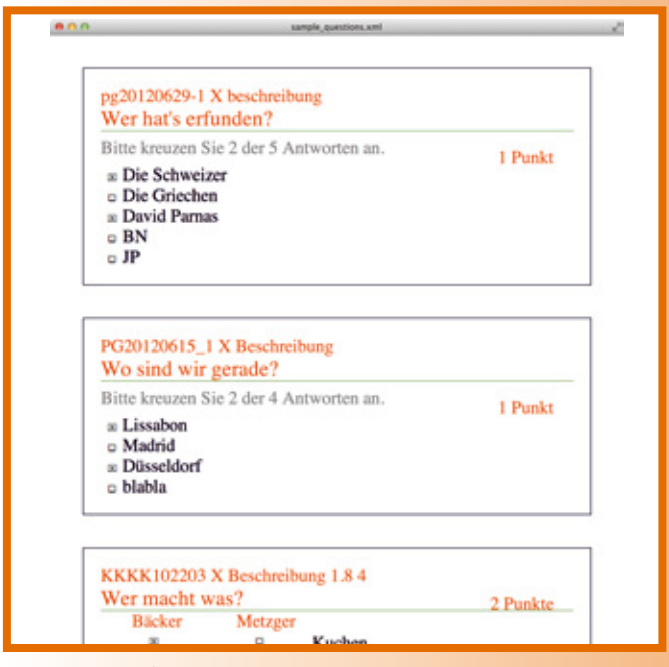

Abb. 3: Bildschirmfoto des XML-Dokuments – mit Stylesheet-Anweisungen

Solche Processing-Instructions versteht unser Browser direkt und kann mit diesen ein XML-Dokument sehr ansehnlich – wie in Abbildung 3 dargestellt – rendern. Ohne Processing-Instructions würde das Dokument wie in Abbildung 2 dargestellt. Ein riesiger Vorteil der Verwendung solcher Stylesheet-Angaben im XML ist, dass diese auch ohne von uns entwickelte Software von vielen Browsern richtig interpretiert werden.

Die Bildschirmfotos in den Abbildungen 2 und 3 zeigen jeweils das gleiche XML-Dokument. Einmal sind die Stylesheet-Anweisungen auskommentiert – und einmal nicht.

Sollte Ihr Browser die Seite [questionnaire] in etwa wie in Abbildung 3 darstellen, unterstützt er das Rendern von XML mit Stylesheet-Anweisungen.

XSLT für Ergänzungen

Um mit einem CSS-Sheet XML-Strukturen zum Beispiel tabellarisch darzustellen, benötigen wir im XML jeweils Elemente für die Tabelle, Tabellenzeilen sowie Tabellenzellen.

Nicht immer hat man aber im XML eine für die Darstellung entsprechende Struktur. Dies lässt sich mit einem XSLT-Stylesheet einfach beheben. Das im obigen Beispiel eingebundene Stylesheet enhance-sheet.xslt ergänzt die in Abbildung 2 erkennbaren Optionen um Elemente, die eine Tabellenzeile repräsentieren sollen.

```
...
<xsl:template match="t:question[@kind='K']/t:options/t:a">
  <xsl:element name="response">
    <xsl:element name="checked"/>
    <xsl:element name="unchecked"/>
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:element>
</xsl:template>

<xsl:template match="t:question[@kind='K']/t:options/t:b">
  <xsl:element name="response">
    <xsl:element name="t:unchecked"/>
    <xsl:element name="t:checked"/>
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:element>
</xsl:template>
...
```

Listing 4: Auszug aus enhance-sheet.xslt

Listing 4 zeigt einen Auszug aus dem XSLT-Stylesheet, das für jedes Element im Wesentlichen die XML-Knoten so belässt. Eine Ausnahme bilden die Antwortmöglichkeiten von Fragen eines bestimmten Typs. Das Listing zeigt, wie mit den Optionen a und b der Fragen vom Typ K verfahren werden soll: Statt einen solchen Knoten a (bzw. b) einfach zu übernehmen, werden drei neue Knoten erzeugt. Aus

```
<a>Kuchen</a>
```

wird mit dem XSLT

```
<response>
  <checked/>
  <unchecked/>
  <a>Kuchen</a>
</response>
```

Die resultierende Struktur lässt sich per CSS nun einfach optisch aufbereiten.

CSS für XML

CSS lässt sich für die Darstellung von XML-Elementen genau so verwenden, wie für HTML. Listing 5 zeigt einen Auszug der CSS-Datei, die vom obigen XML-Dokument referenziert wird.

Zu sehen sind die Format-Definitionen für die CSS-Selektoren `a`, `b`, `checked`, `response` und `unchecked`. Dank der vorigen XSLT-Transformation kann das Layout nun einfach angegeben werden: `response` soll als table-row und die anderen Elemente sollen als table-cell dargestellt werden. Das Ergebnis ist am unteren Bildrand von Abbildung 3 erkennbar.

```
response {
  display: table-row;
}
unchecked, checked {
  display: table-cell;
  text-align: center;
  background-position: center;
  background-repeat: no-repeat;
  height: 1em;
}
checked {
  background-image: url('...');
}
unchecked {
  background-image: url('...');
}
a, b {
  display: table-cell;
  margin-left: 20px;
}
```

Listing 5: Das vom XML-Dokument referenzierte CSS-Stylesheet questionnaire.css

JAXB – eine große Hilfe?

In vielen Java-Projekten begegnet uns XML in Kombination mit JAXB, der Java Architecture for XML Binding. JAXB ist ein Framework, das XML zum Serialisierungsformat degradiert. Eine kurze Einführung in JAXB bietet beispielsweise Torsten Horns Tutorium [JAXB-Tutorial].

Um mit JAXB Processing-Instructions in ein XML-Dokument einzufügen, die dann die Benutzbarkeit auch außerhalb unserer Softwaresysteme erhöhen, gibt es zwei Alternativen. Entweder können wir zusätzliche Informationen selbst ausgeben, bevor JAXB ein XML-Fragment schreibt, oder aber wir setzen beim Marshaller das Property `com.sun.xml.bind.xmlHeaders` mit dem entsprechenden Preamble-String.

Ich habe bislang JAXB gekapselt, um die Processing-Instructions selbst in die Ausgabe zu schreiben. Listing 6 zeigt exemplarisch eine Hilfsmethode, mit der man die XML-Ausgabe einfach anreichern kann.

```
public static void marshall(
    PrintWriter writer,
    Object theObject,
    String... prependLiterals ) throws JAXBException {
  Marshaller marshaller = JAXBContext.newInstance(
    theObject.getClass() ).createMarshaller();
  marshaller.setProperty( Marshaller.JAXB_FRAGMENT, Boolean.TRUE );
  marshaller.setProperty(
    Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE );
  writer.println("<!-- marshall(...) - begin ---------->");
  for ( String s : prependLiterals ) {
    writer.println( s );
  }
  marshaller.marshal( theObject, writer );
```

```
    writer.println("\n<!-- marshall(...) - end ------------------->");
    writer.flush();
  }
```

Listing 6: Die Methode marshall() kapselt JAXB und ermöglicht, beliebige Zeichenketten in den XML-Output einzufügen

Damit haben wir den ersten Schritt für die Erzeugung selbstbeschreibender Dokumente getan. Der folgende Aufruf erzeugt dann das gewünschte Ergebnis:

```
marshall(
  theWriter,
  questions,
  "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>",
  "<?xml-stylesheet type=\"text/xml\" href=\"./enhance-sheet.xslt\" ?>",
  "<?xml-stylesheet type=\"text/css\" href=\"./questionnaire.css\" ?>" );
```

Bleibt eine Einschränkung: Das Repräsentieren von Kommentaren im XML ist nicht in JAXB vorgesehen. Daher sind Informationsverluste beim De- und erneuten Serialisieren nicht auszuschließen.

Fazit

XML ist bekanntermaßen eine mächtige Beschreibungssprache, mit der Dokumente selbsterklärend beschrieben werden können. Die XML-Unterstützung ist sowohl in den gängigen Programmiersprachen als auch in gängigen Webbrowsern gut. Die flexible Verarbeitung von XML erfordert eigentlich keine zusätzlichen Bibliotheken über die im Java SDK enthaltenen Bestandteile hinaus. Mit Hilfe weniger „Hilfsstrukturen" ist es möglich, Werte aus dem XML zu lesen oder mittels Transformationen zu setzen.

Binding-Frameworks degradieren XML-Dokumente häufig zu „Dateien" mit serialisierten Objekten. In dieser Beziehung hat das einfachere und kompaktere JSON-Format einige Vorteile aufgrund der geringen Größe und der direkten Interoperabilität mit JavaScript.

Wer allerdings XML-Dokumente für eine Reihe von Nutzern mit unterschiedlichen Werkzeugen exportieren will, profitiert von den Möglichkeiten in XML Dokumenttyp, Angaben zur Darstellung und die Nutzdaten zu trennen, aber für alle Werkzeuge verständlich zu verknüpfen.

Insbesondere die Stylesheet-Anweisung ermöglicht Browsern, XML-Dokumente leserlich aufzubereiten.

Links

[CSS-Spec] Cascading Style Sheets (CSS) Snapshot, `http://www.w3.org/TR/css-2010/`
[JAXBTutorial] `http://www.torsten-horn.de/techdocs/java-xml-jaxb.htm`
[Krüg02] Ch. Krüger, Einführung in JAXB, in: JavaSPEKTRUM, 5/2002
[questionnaire] `http://ghadir.de/questionnaire/sample1.xml`
[XMLSpec] XML Specification Version 1.0, `http://www.w3.org/TR/xml/`
[XSLTSpec] XSL Transformations 1.0, `http://www.w3.org/TR/xslt/`

Phillip Ghadir baut am liebsten tragfähige, langlebige Softwaresysteme. Er ist Mitglied der Geschäftsleitung bei innoQ und hat sich früh auf Architekturen für verteilte, unternehmenskritische Systeme spezialisiert. Darüber hinaus ist er Mitbegründer und aktives Mitglied des iSAQB, des International Software Architecture Qualification Board.
E-Mail: phillip.ghadir@innoq.com