XML, FI und JSON strukturiert in Java verwenden

Wolfgang Nast

Agenda

- Java und
  - XML
  - FastInfoset
  - JSON
- Zusammenfassung

Im Überblick

Technologie-orientiert
Branchen-unabhängig

Hauptsitz
Ratingen

240
Beschäftigte

MT AG

Zertifizierter
Partner von
Oracle,
Microsoft
und SAP

Inhabergeführte
Aktiengesellschaft

Gründungsjahr
1994

Ausbildungs-
betrieb

kununu
TOP COMPANY
www.kununu.com
VON MITARBEITERN EMPFOHLEN!

Niederlassung
Frankfurt am Main

MT AG

Vorstellen der Technologien

MT AG

Java und XML

- XML lesen mit SAX
- XML lesen und schreiben mit DOM
- XML lesen und schreiben mit StAX
- XML lesen und schreiben mit JAXB

XML lesen mit SAX

Strukturiert die Daten sammeln

- Als Events lesen:
  - DefaultHandler für die Events anpassen
  - switch für Elementnamen in Start und Ende
  - Inhalt sammeln und zuordnen
  - Daten konvertieren `(<,>,',",&)`

MT AG

XML lesen mit SAX

Einfaches Beispiel Teil 1

- Als Datei einlesen:

```
SAXParser parser =
SAXParserFactory.newInstance().newSAXParser();
XMLReader saxReader = parser.getXMLReader();
saxReader.setContentHandler(new LeseSAX());
saxReader.parse(new InputSource(new
FileInputStream("Beispieldatei")));
```

XML lesen mit SAX

Einfaches Beispiel Teil 2

- Als DefaultHandler überscheiben:

```
@Override
public void startElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException {
   switch (localName) {
      case "Element":
         //Element verarbeiten
         break;
   }
}
```


MT AG

XML lesen mit SAX

Stärken und Schwächen

- Stärken:
  - Eventbasiert
  - wenig Speicher
  - validieren gegen Schema
- Schwächen
  - selber Daten wandeln und strukturieren
  - keine aktives weiterbewegen
  - kein Schreiben

XML lesen mit DOM

Strukturiert die Daten Sammeln

- Als Dokument lesen
  - Daten als Document
  - Textnodes und CData zusammenfassen
  - keine Ersatzzeichen verwendet
  - DOM Navigation bekannt aus JavaScript

XML lesen mit DOM

Einfaches Beispiel

- Als Datei einlesen:

```
Document db = DocumentBuilderFactory.newInstance()
    .newDocumentBuilder().parse(new
    FileInputStream("Beispiel.xml"));
    //hier Inhalt auslesen
```

XML schreiben mit DOM

Strukturiert die Daten aufbauen und schreiben

- Als Dokument erstellen
  - Daten als Document
  - DOM Struktur erstellen
  - Transformer verwenden zum Schreiben

MT AG

XML schreiben von DOM mit Transformer

Einfaches Beispiel

- Als Datei schreiben:

```
Document doc = … //Document erstellen

Transformer tr = TransformerFactory
    .newInstance().newTransformer();

tr.transform(new DOMSource(doc), new StreamResult(
    new File("Ergebnis.xml")));
```

XML lesen und schreiben mit DOM

Stärken und Schwächen

- Stärken:
  - Strukturiert als Document
  - Textnodes schon richtig
  - Validieren gegen Schema
- Schwächen
  - Umständliche Struktur
  - Hoher Speicherverbrauch
  - Schreiben mit Transformer

XML lesen mit StAX

Strukturiert die Daten lesen, zwei Möglichkeiten

- Als Stream lesen
  - Daten als Stream (Events, direkt)
  - Switch für Type und Elementnamen
  - Gezieltes lesen möglich
  - Element wieder zurückstellen(Event)

XML lesen mit StAX

Struckturiert die Daten sammeln

- Als Stream lesen:

```
XMLStreamReader streamRd =
XMLInputFactory.newFactory().createXMLStreamReader(new
FileInputStream("Beispiel.xml"));
```

- Als Events lesen:

```
XMLEventReader eventRd = XMLInputFactory.newFactory()
.createXMLEventReader(new FileInputStream("Beispiel.xml"));
mlStreamReader xr = fac.createStreamReader();
```

XML schreiben mit StAX

Strukturiertes Schreiben

- Als Stream schreiben:
  - Daten als Stream (Events, direkt)
  - Ereignisweise schreiben
  - Endelement scheiben optional
  - Kopieren des Rests(Event)

XML schreiben mit StAX

Strukturiert die Daten schreiben

- Als Stream schreiben:

```
XMLStreamWriter streamWr = XMLOutputFactory.newFactory()
.createXMLStreamWriter(new
FileOutputStream("Beispiel.xml"));
```

- Als Events schreiben:

```
XMLEventWriter eventWr = XMLOutputFactory.newFactory()
.createXMLEventWriter(new FileOutputStream("Beispiel.xml"));
```

XML lesen und schreiben mit StAX

Stärken und Schwächen

- Stärken:
  - Elementweise lesen
  - wenig Speicher
  - kombinierbar mit JAXB
  - Schreiben
- Schwächen
  - kein Validieren
  - Namespaces umständlich

XML Schema und JAXB Context

Vorbereiten erstellen der Java Klassen

- Java Klassen generieren:

`xjc` verwenden um aus dem XML Schema Java Code zu generieren.

- JAXBContext initialisieren:

```
JAXBContext cont1 = JAXBContext.newInstance("packageliste");
JAXBContext cont2 = JAXBContext.newInstance(
   GenerierteKlasseA.class, GennerierteKlasseB.class);
```

XML lesen und scheiben mit JAXB

- Als Java Objekt lesen:
  - JAXBContext erstellen
  - Mit Unmarschaller lesen
- Als Java Objekt schreiben:
  - JAXBContext erstellen
  - Mit Marshaller schreiben

XML lesen und schreiben mit JAXB

- Mit Unmarshaller lesen:

```
Object obj = cont.createUnmarshaller().unmarshal(new
FileInputStream("Beispiel.xml"));
```

- Mit Marshaller schreiben:

```
cont.createMarshaller().marshal(obj, new
FileOutputStream("Beispiel.xml"));
```

XML lesen und schreiben mit JAXB

Stärken und Schwächen

- Stärken:
  - Komplettes Dokument oder Fragment im Speicher
  - strukturierter Speicherverbrauch moderat.
  - Base64, Hex <-> byte[]
  - Schreiben
- Schwächen
  - Generierter Code
  - JAXBContext braucht Resourcen

Java und FastInfoset

- FI lesen mit SAX, DOM und StAX
- FI schreiben mit DOM und StAX
- FI lesen und schreiben mit JAXB

FastInfoset lesen mit SAX, DOM und StAX

- Gleiche Strukturen wie bei XML:
  - SAX Parser (Events)
  - DOM Document
  - StAX Reader (Stream)

FastInfoset lesen mit SAX, DOM und StAX

- Lesen mit Konstruktoren anstelle von Factory:

```
XMLReader saxFi = new SAXDocumentParser();

Document doc = DocumentBuilderFactory.newInstance()
    .newDocumentBuilder().newDocument();

new DOMDocumentParser().parse(doc, new
FileInputStream("Beispiel.fi"));

XMLStreamReader xre = new StAXDocumentParser(new
FileInputStream("Beispiel.fi"));
```

FastInfoset lesen mit SAX, DOM und StAX

Stärken und Schwächen

- Stärken:
  - gleiche Strukturen
  - Base64 <-> byte[] (StAX)
- Schwächen
  - leeres Dokument(DOM)
  - kein XMLEventReader(StAX)
  - leicht geänderte Befehle

FastInfoset schreiben mit SAX, DOM und StAX

- Gleiche Strukturen wie bei XML:
  - DOM Serialiser
  - StAX Serialiser (Stream)

XML schreiben mit DOM und StAX

- Schreiben mit Konstruktoren anstelle von Factory:

```
DOMDocumentSerializer docSer = new DOMDocumentSerializer();
docSer.setOutputStream(new FileOutputStream(
   "Beispiel.fi"));
docSer.serialize(doc);
XMLStreamWriter xwr = new StAXDocumentSerializer(new
FileOutputStream("Beispiel.fi"));
```

FastInfoset schreiben mit DOM und StAX

Stärken und Schwächen

- Stärken:
  - gleiche Strukturen
  - Base64<-> byte[] (StAX)
  - direktes Schreiben(DOM)
- Schwächen
  - kein XMLEventWriter(StAX)
  - leicht geänderte Befehle

FastInfoset lesen und schreiben mit JAXB

Verwendung von StAX

- JAXB lesen:
  - mit SAX
  - mit DOM
  - mit StAX
- JAXB schreiben
  - mit DOM
  - mit StAX

FastInfoset lesen und schreiben mit JAXB

Stärken und Schwächen

- Stärken:
  - Verwendung der XML, FI Technologien
  - empfohlen StAX lesen und schreiben
  - Base64 wird byte[]
- Schwächen
  - keine eigenen Reader und Writer
  - nicht SAX Standard, wie bei XML

Java und JSON

- JSON lesen und schreiben Java JSON
- JSON lesen und schreiben mit REST API

JSON lesen (JavaEE 7)

Strukturiert die Daten lesen, zwei Möglichkeiten

- Als Stream lesen
  - Daten als Stream
  - Switch für Type und Objektnamen
  - Gezieltes lesen möglich
- Als Struktur lesen
  - Daten als Struktur
  - Navigieren über die Struktur

JSON lesen

Struckturiert die Daten sammeln

- Als Stream lesen:

```
JsonParser jsonParser = Json.createParser(new
FileInputStream("Beispiel.json"));
```

- Als Struktur lesen:

```
JsonStructure jsonStruct = Json.createReader(new
FileInputStream("Beispiel.json")).read();
```

JSON schreiben (JavaEE 7)

Strukturiertes Schreiben

- Als Stream schreiben
  - Daten als Stream
  - Elementweise schreiben
- Als Struktur schreiben
  - Daten als Struktur schreiben
  - Erstellen der Struktur mit Buildern

JSON schreiben

Strukturiert die Daten schreiben

- Als Stream schreiben:

```
JsonGenerator jsonWr = Json.createGenerator(new
FileOutputStream("Beispiel.json"));

  jsonWr.writeXXX(YYY);//Struktur und Daten
```

- Als Struktur schreiben:

```
JsonObject job = Json.createObjectBuilder().add("Wert1",
"ein Wert").add("Auswerten", true).add("Parmeter",
Json.createArrayBuilder().add(3).add(4)).build();
Json.createWriter(new
FileWriter("Beispiel.json")).writeObject(job);
```


MT AG

JSON lesen und schreiben (JavaEE 7)

Stärken und Schwächen

- Stärken:
  - Elementweise lesen
  - Struktur lesen
  - wenig Speicher
  - Builder für Struktur
- Schwächen
  - eigene API
  - Struktur braucht mehr Speicher

JSON mit REST API (JAX RS)

Verwenden von JAXB Klassen

- JSON liefern als Antwort:

```
@GET
@Produces("application/json")
public String getClichedMessage() {
    return jaxbObject;
}
```

- JSON entgegen nehmen:

```
@POST
@Consumes("application/json")
    public void postClichedMessage(XmlObject jaxbObject) {
```

REST API (JAX RS) Mime Types

- Unterstützte Mime Types JSON
  - application/json
  - text/json
- Unterstützte Mime Types XML
  - application/xml
  - text/xml

JSON lesen und schreiben REST(JAX RS)

Stärken und Schwächen

- Stärken:
  - Java Struktur entgegennehmen
  - Java Struktur liefern
  - JAXB für XML und JSON
- Schwächen
  - nicht Standart (Jersey und RESTEasy)
  - benötigt JAXB

MT AG

data . cloud . mobile

ROADSHOW
„VON FORMS NACH APEX“

24.01.2017 Hamburg
25.01.2017 Düsseldorf
26.01.2017 Frankfurt
27.01.2017 München

Wolfgang Nast
Telefon: +49 2102 30961 – 0
wolfgang.nast@mt-ag.com