Speichersparende XML Verarbeitung mit StAX und JAXB

Wolfgang Nast
MT AG
Ratingen

Schlüsselworte

Java XML StAX JAXB Optimierung

Einleitung

Es wird vorgestellt, wie durch die Kombination der beiden Technologien JAXB und StAX ein Speicher effizientes arbeiten möglich ist, wobei weiterhin ein strukturiertes zugreifen auf XML Teile möglich ist. Dazu werden zuerst StAX und JAXB einzeln vorgestellt und im Anschluss die Kombination beider.

JAXB

JAXB dient zum objektorientierten Abbilden von XML Strukturen auf Java Strukturen. Dafür wird das XML Schema benötigt. Mit dem Befehl

```
xjc <Schema.xsd>
```

werden die Klassen für Java generiert. Um auf die Klassen zuzugreifen wird der Context benötigt. Dieser wird mit

```
JAXBContext cont = JAXBContext.newInstance(<BasisElement>);
```

geholt. Zum Lesen einer XML Datei wird der Unmarshaller verwendet. Mit

```
Unmarshaller Umar = cont.createUnmarshaller();
```

wird ein neuer Unmarshaller geholt. Mit

```
BasisElement xmldaten = (BasisElement)umar.unmarshall("Dateiname.XML");
```

wird die XML Datei eingelesen und in die Java Struktur überführt. Jetzt können die interessanten Werte ausgelesen und bearbeitet werden. Nach dem Bearbeiten kann man die XML Struktur mit Hilfe des Marshallers schreiben. Dafür ist der Marshaller mit

```
Marshaller mar = cont.createMarshaller();
```

anzulegen und mit

```
mar.marshall("NeuerDateiname.xml");
```

zu schreiben. Soll eine andere Struktur geschrieben werden, so ist mit xjc diese auch zu generieren und der zweite Context zu laden.

Der Vorteil von JAXB:

Alles ist in der Java Struktur vorhanden und das XML kann mit einem Befehl gelesen oder geschrieben werden.

Der Nachteil von JAXB:

Alles muss im Speicher sein und beim Transformieren wird noch mehr Speicher benötigt. Dies führt zu Problemen, wenn die XML Dateien größer werden.

StAX

StAX ist dafür gedacht, sich die XML Elemente und Anweisungen einzeln nach einander abzuholen. Es gibt zwei Implementierungen, die Objektorientierte XMLEventReader und XMLEventWriter. Dabei werden alle XML Daten als Objekt mit seinen Attributen abgebildet. Dies gilt besonders bei dem StartElement, wo alle Attribute und Namespaces Teil des Objektes sind. Die zweite Implementierung XMLStreamReader und XMLStreamWriter ist funktional umgesetzt, was meist praktischer ist, weil man an die Daten im XML möchte, wozu man keine Objekte benötigt. Hier navigiert man entweder gezielt zudem bekannten Daten oder man sucht abhängig von dem Datentyp und den gewünschten Eigenschaften gezielt zu dem Element oder Daten. Es werden nur die interessanten Stellen im XML angesteuert, der Rest wird einfach überlesen.

Der Vorteil von StAX ist, das man gezielt zu den Daten navigiert, die Interessant sind.

Der Nachteil von StAX ist, das man muss alle Daten in einer Struktur die man benötigt einzeln ansteuern, welches recht aufwendig werden kann.

JAXB und StAX

Zur besseren Nutzung von der XML Verarbeitung werden die Vorteile beider Verfahren kombiniert. Das heißt in der Praxis: Mit StAX wird zum XML Fragment navigiert, welches verarbeitet werden soll. Das ganze Fragment wird mit JAXB eingelesen. Beim Schreiben muss die äußere XML Struktur mit StAX geschrieben werden, da nicht alles im Speicher gehalten werden soll(bei JAXB).
Das ist manchmal recht mühselig, aber nicht zu vermeiden, wenn dieser Teil keine Fragmente enthält. Die Fragmente werden dann mit JAXB geschrieben.

Genug Theorie, jetzt kommt ein praktisches Beispiel um zu Zeigen, wie Daten verarbeitet werden können.

Zuerst die Strukturen in XML mit den Schemata:

Quelle :

Mitarbeiterhirachie mit Chef, Abteilungsleitern und Mitarbeitern.
Metainformation: Stand der Aufstellung, Anzahl Abteilungen und Summe aller Mitarbeiter ohne Chef und Abteilungsleiter.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:tns=http://www.beispiel.de/2014/MitarbeiterHirachie

targetNamespace=http://www.beispiel.de/2014/MitarbeiterHirachie
        elementFormDefault="qualified">
    <element name="Firma" type="tns:FirmaType"/>
    <complexType name="FirmaType">
        <sequence>
            <element name="MetaDaten" type="tns:MetaDatenType"/>
            <element name="Chef" type="tns:PersonType"/>
            <element name="Abteilung" type="tns:AbteilungType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="MetaDatenType">
        <attribute name="Stand" type="date" use="required"/>
        <attribute name="AnzahlMitarbeiter" type="long"
use="required"/>
        <attribute name="AnzahlAbteilungen" type="long"
use="required"/>
    </complexType>
    <complexType name="AbteilungType">
        <sequence>
            <element name="Abteilungsleiter" type="tns:PersonType"/>
            <element name="Mitarbeiter" type="tns:PersonType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="PersonType">
        <attribute name="Nachname" type="string" use="required"/>
        <attribute name="Vorname" type="string" use="required"/>
        <attribute name="Eintrittsdatum" type="date"
use="optional"/>
    </complexType>
</schema>
```

Ziel:
Personen ohne Hierarchie.
Metainformation: Anzahl an Personen und Stand der Aufstellung.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns=http://www.w3.org/2001/XMLSchema
        xmlns:tns=http://www.beispiel.de/2014/Mitarbeiter
        targetNamespace=http://www.beispiel.de/2014/Mitarbeiter
        elementFormDefault="qualified">
    <element name="MitarbeiterListe" type="tns:MitarbeiterType"/>
    <complexType name="MitarbeiterType">
        <sequence>
            <element name="MetaDaten" type="tns:MetaDatenType"/>
            <element name="Mitarbeiter" type="tns:PersonType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="MetaDatenType">
        <attribute name="Stand" type="date" use="required"/>
        <attribute name="Anzahl" type="long" use="required"/>
    </complexType>
    <complexType name="PersonType">
        <attribute name="Nachname" type="string" use="required"/>
        <attribute name="Vorname" type="string" use="required"/>
        <attribute name="Eintrittsdatum" type="date"
use="optional"/>
        <attribute name="Leiter" type="boolean" default="false"/>
    </complexType>
</schema>
```

Zusammenfassung:

Es ist leicht möglich die Vorteile von StAX und JAXB zu kombinieren. Hierbei ist es notwendig immer eine sinnvolle Balance zwischen Speicherverbrauch, vorhalten von Daten und dem direktem Speichern der Daten zu finden. Sind die Daten recht klein, so kann alles mit JAXB direkt verarbeitet werden. Werden nur gezielt wenige Daten gelesen oder verändert so ist StAX alleine von Vorteil. Müssen viele Daten auf Vorrat gehalten werden, weil sie neu sortiert ausgegeben werden sollen, so kann es immer noch zu Problemen im Speicher führen. Dann muss überlegt werden, wie man die Daten auslagert oder zwischen speichert, als Datei oder in einer Datenbank. Wichtig ist zu sehen, wie sich die beiden Technologien bei großen Datenmengen gut ergänzen und einem die Arbeit erleichtern.

Kontaktadresse:

Wolfgang Nast

MT AG

Balcke-Dürr-Allee 9
D-40882 Ratingen

Telefon:	+49 (0) 2102 30961-0
Fax:	+49 (0) 2102 30961- 101
E-Mail	Wolfgang.Nast@mt-ag.com
Internet:	www.mt-ag.com