



JAXB 2.0 How to

//@author: Peter Huster



Agenda

1. Requirements
2. JAXB-API
3. XML Schema to Java
4. Java to XML Schema
5. Quellen



1. Requirements

Überblick

- Spezifiziert nach JSR 222
- Annotationsgetriebenes Framework
- Schema-Compiler
- Schema-Generator
- Binding Framework

3

Hervorgegangen aus JCP mittlerweile Bestandteil von Java (6)

Java Klassen werden mit jeder Menge an Annotation bestückt um JAXB mit Informationen zu füttern

Schema-Compiler kann aus existierenden Schemas das Java Klassen Modell generieren

Schema-Generator bindet ein existierendes Java Datenmodell an ein generiertes Schema

Das Binding Framework besteht primär aus der API zum Marshalling/Unmarshalling, Validierung und dem Binder



1. Requirements

Alternativen zu JAXB

- XML Binding
 - XMLBeans
 - JiBX, JaxMe2...
- XML Processing
 - sax, stax
 - Dom4j

4

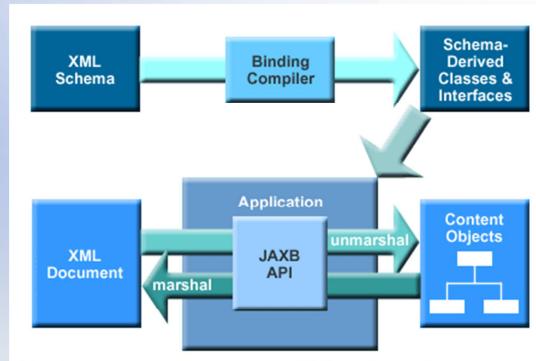
Alternativ kann man XMLBeans nutzen welches sich ähnlich handhabt wie JAXB aber eine mächtigere Navigation bietet. Nachteil ist das es keinen Binder gibt und es nicht Bestandteil von Java ist (sondern apache ins leben gerufen wurde und damit extra eingebunden werden muss... wie JAXB vor Java 6)

Viele weitere Binding Frameworks wie JiBX nutzen das Prinzip des Bindens sind aber i.d.R. weniger angenehm zu handhaben

Oder man greift auf XMLProcessing zurück die natürlich performanter sein können wenn das Parsen effizient umgesetzt wurde, dafür büßt man ungemein an Komfort ein

1. Requirements

Architektur – Wie funktioniert XML Binding



5

Der obere Abschnitt bestehend aus XML-Schema, Binding Compiler und Schema-Derived Classes und Interfaces zeigt wie der Applikation die POJOs zur Verfügung gestellt werden. Der SchemaCompiler nimmt das Schema und wenn vorhanden Bindingdeklarationen und generiert daraus das Klassenmodell. Natürlich kann man auch ohne Schema arbeiten dann wird selbiges on the fly vom Framework erstellt das Prinzip bleibt aber das gleiche.

Sobald dem Framework die XML Datei zur Verfügung gestellt wird können mit den Schemainformationen die Daten unmarshalled (von XML zu Java) marshalled(von Java zu XML) oder mittels Binder gebunden werden (POJOs werden an die XML Daten gebunden und können so adhoc Daten austauschen



1. Requirements

Installation (für Java 5)

- ANT einbinden
- Aktuelles JAXB RI
 - bin: Scripte für Schema-Compiler/Generator
 - docs: Dokumentation
 - lib: Bibliotheken + Quellcode Archive
 - samples: Beispiele
- Bibliotheken einbinden

6

Falls Eclipse genutzt wird erledigt sich der erste Punkt von selbst. Die aktuelle Referenzimplementierung kann bei Sun geladen werden. Es gibt auch JWSDP in dem JAXB2.0 schon eingebunden ist, aber nicht mit der aktuellen RI.

Die zu bindenden Bibliotheken

jaxb-api.jar: API Klassen der Spezifikation

jaxb-impl.jar: Klassen der JAXB-RI

jsr173_1.0_api.jar: XML Streaming API

activation.jar: Abhängigkeit zur Java Beans Activation API

jaxb-xjc.jar: XJC Schema-Compiler und Schema Generator

Mit den Libs im CLASSPATH ist man auf der sicheren Seite (wenn man mit Eclipse arbeitet kann man auch Quellcode Archive in den Klassenpfad aufnehmen damit man die JavaDoc immer dabei hat)

Laut Dokumentation stehen die Anttasks des xjc bei Java 6 nicht zur Verfügung, also sollte man selbige via der RI einbinden



1. Requirements

JAXB Dictionary

- Mapping-Annotation
- Bindungskonfiguration
- Marshalling
- Unmarshalling
 - Flexible unmarshalling
 - Structural unmarshalling
- Binder

7

Mapping Annotationen sind Annotationen nach JSR 175 die an Java Klassen gebunden werden. Zum Beispiel @XmlRootElement um dem Framework die Information zur Verfügung zu stellen das die annotierte Klasse das XML Wurzelement ist.

Alternativ kann man den Schemagenerator mit den annotierten Klassen füttern sodass er ein passendes Schema generiert

Bindungskonfigurationen sind Annotationen im XML Schema das heißt diese werden innerhalb des annotations-Tags erstellt und damit an das Schema Element angehängt

Der Schemacompiler schnappt sich damit die Schema Dateien und die passenden Bindungskonfigurationen und kann daraus die Java Klassen erstellen

Marshalling ist das Speichern von XML d.h. aus den vorhandenen Daten die als POJOs existieren wird eine persistente XML Datei generiert

Unmarshalling ist das Laden von XML d.h. die persistenten Daten der XML Datei werden in die POJOs geladen

Flexibles unmarshalling wendet sich an die ständige Evolution von Projekten und transformiert die Elemente nach ihrem Namen, statt nach der Position (ermöglicht das arbeiten mit Dokumenten in denen die Reihenfolge nicht übereinstimmt, Elemente unbekannt oder fehlend sind)

Structural Unmarshalling ist das strikte Unmarshalling von XML Dokumenten und wirft Fehler sobald Abweichungen in der Reihenfolge auftreten. Das unmarshalling basiert auf der Struktur des XML Dokuments und war die Grundlage der JAXB1.0 Spezifikation



1. Requirements

Vorausgesetzte Technologien

- XML
- XML Schema
- XPath
- ANT

XML und XML Schema setzt einen Verweis auf die hervorragende Präsentation von Gerrit Beine.

Desweiteren sollte man wissen was ein offenes Schema ist wie man mit Namespaces, Vererbung, Kardinalität und Eindeutigkeit von Elementen umgeht.

Traversieren mit XPath ausdrücken und Ant Grundlagen sind „nice to haves“ um sich das JAXB Leben einfacher zu Gestalten



2. JAXB API

Hallo JAXB - Einführungsbeispiel

- Java Repräsentation des HelloWorld Tags

```
@XmlRootElement  
Public class HelloWorld{  
private String _message;  
public String getMessage(){return message;}  
public void setMessage(String message){  
    _message=message;  
}  
}
```

Beispiel Code für ein HelloWorld element. Man Beachte die
@XmlRootElement annotation



2. JAXB API

Hallo JAXB - Einführungsbeispiel

- Verwenden der HelloWorld Klasse

```
Public class Beispiel{  
    Public static void main(String args[]) throws Exception  
    {  
        JAXBContext mycontext = JAXBContext.newInstance(HelloWorld.class);  
        Marshaller mymarshaller = mycontext.createMarshaller();  
        mymarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,true);  
        HelloWorld hello = new HelloWorld();  
        hello.setMessage(„Hallo JAXB“);  
        mymarshaller.marshal(hello, System.out);  
    }  
}
```

10

Beispiel welches die HelloWorld Klasse nutzt.

Wir erstellen einen BindingContext anhand der HelloWorld Klasse. Im Hintergrund bastelt sich JAXB aus diesen Informationen ein Schema gegen welches dann validiert wird (klingt komisch, aber JAXB setzt immer ein Schema voraus)

Mittels des Contextes kann ein Marshaller erstellt werden der dann das POJO in XML wandelt

Die zusätzlichen Properties sind nur der Schönheit halber aktiviert

Zu guter letzt Instanziieren wir ein Objekt und füllen es mit Daten welches wir dann dem Marshaller übergeben damit er es ausgeben kann



2. JAXB API

JAXB Grundlagen

- JAXBContext
 - XML read/write Voraussetzung
 - kann erzeugt werden: via Paketname, mit anderem Classloader, via Java Klassen
- JAXBIntrospector
 - zur Identifikation von gebundenen Elementen
- ObjectFactory
 - stellt JAXBElements her

11

JAXBContext wird über newInstance erzeugt und bietet damit implizit den read/write Zugriff auf XML

Erzeugt man den Context mit einem String erwartet er Paketnamen die die zu bindenden XML Schemas repräsentieren

Will man (warum auch immer) einen speziellen Classloader ins Boot holen gibt man den einfach dazu an

Den Context mit einer Klasse zu erzeugen setzt voraus das die Klasse die Annotation @XmlRootElement vorweisen kann, da es ja theoretisch nur ein Wurzelement geben kann, bezieht der Context daraus alle Informationen

Clever ist es den Context als Konstante zu definieren (bzw. das Instanzieren des Contextes nur einmal stattfinden zu lassen) um das ganze etwas performanter zu gestalten

JAXBIntrospector ist ein Helfer der unbekannte Java Objekte auf XML Binding prüfen kann mit isElement

Die ObjectFactory wird vom SchemaCompiler erzeugt und kann mit ihren create Methode

z.B. mit ObjectFactory fabrik = new ObjectFactory();

TolleKlasse klasse = fabrik.createTolleKlasse();

Will man ein JAXBElement (z.B. um Namespace Informationen hinzuzufügen) erzeugen nutzt man einen Wrapper

JAXBELEMENT mywrapper = fabrik.createTolleKlasse(klasse);



2. JAXB API

Marshalling – Java zu XML

- Transformiert Java Objekte in XML
- wird über den Context instanziert
- kann in verschiedene Ausgabeformate transformieren(Dateien, Streams, DOM, stax...)
- mittels Properties kann der Prozess angepasst werden
- auch ungültige Inhalte können transformiert werden

12

Siehe eingehendes Beispiel und Doku.

Will man beliebige Objekte marshallen ist man auf den JAXBElement Wrapper angewiesen, da der Marshaller normalerweise die @XMLElement Annotation voraussetzt



2. JAXB API

Unmarshalling – XML zu Java

- transformiert XML in Java Objekte
- verarbeitet File, URL, Inputstream, TrAX, DOM, SAX, XMLStreamreader, XMLEventreader
- mit any Elementen wird flexibles Unmarshalling realisiert
- mit JAXBELEMENT können Teilbäume und nicht Wurzelemente unmarshalled werden
- mittels xsi:type Element können Elemente auf anderen Elementen abgebildet werden

13

Der Unmarshaller transformiert XML in Java Objekte und verarbeitet dabei verschiedenste Eingabeformate

Über den Context kann eine Instanz des unmarshallers erzeugt werden, akzeptiert dabei aber keinerlei Konfigurationsmöglichkeiten (wie der marshaller)

Will man Elemente ohne @XMLRootElement verarbeiten benötigt man einen Workaround über JAXBELEMENT

Das gleich gilt wenn man nur ein Stück des Baumes unmarshallen will

Findet eine Evolution der XML Daten statt kann mittels any Element flexibles Unmarshalling realisiert werden



2. JAXB API

Non-Rootelements unmarshalen

```
...
JAXBElement<include> element =
(JAXBElement)unmarshaller.unmarshall (new File(„myIncludeFiles.xml“));
Include include = (Include) element.getValue();
...
...
```

14

Hier umgeht man das Problem wenn man kein globales Element zur Verfügung gestellt bekommt aber die Daten trotzdem verarbeiten will.

Die Anforderung röhrt von der Eindeutigkeit der Daten da der Marshaller sonst nicht wüsste wie er damit umgehen soll



2. JAXB API

Teilbäume unmarshalen

```
...
//Wir erstellen uns ein dom Document mittels Builder dem die XML
//Datei zur Verfügung gestellt wird, der Xpath ausdruck nimmt das document
Node includeNode =
XPathNodeLocator.getNodeUsingXPath(document, "//include");
JAXBElement element =
(JAXBElement)unmarshaller.unmarshall(includeNode,Include.class);
Include include=(Include) element.getValue()
...
...
```

15

Ähnlich zum vorrigen Beispiel wird hier aber auf Basis eines Teilbaumes + expliziter Klassenangabe unmarshalled

Beim Xpath ausdruck ist der Namespace mit anzugeben (hier nicht zu sehen) mittels einer Klasse die NamespaceContext implementiert



2. JAXB API

Validieren von JavaBeans und XML

- Setzt einheitlich beim Marshalling/Unmarshalling ein
- XML Dokumente und Java Objektgraphen können mittels Schema validiert werden
- über ValidationEventHandler kann die Validierung angepasst werden
- ValidationEventCollector registriert mehrere EventHandler
- ungültige Inhalte können weiterverarbeitet werden
- Validierung kann deaktiviert werden

16

Validierung findet beim Marshalling sowie beim Unmarshalling statt

Über setSchema() kann das Schema beispielweise an den unmarshaller gebunden werden

Mit setEventhandler() können Eventhandler beim unmarshaller registriert werden



2. JAXB API

Integration von Applikationslogik

- externen Listener implementieren
- Callback auf Klassenebene definieren
- integriert Applikationsspezifische Logik ins Marshalling/Unmarshalling
- vermeiden von komplexer Logik aufgrund von Performancegründen

17

Die Methoden beforeMarshal, afterMarshal und beforeUnmarshal, afterUnmarshal können im Listener bzw. direkt in der Java Bean angegeben



2. JAXB API

Binder – Vermitteln zwischen DOM und Java

- erzeugt gleichzeitig 2 Sichten auf ein XML Dokument
- Änderungen werden mit den jeweiligen Methoden der Sicht synchronisiert
- der Binder kennt die Beziehungen der Elemente und macht diese nach Außen verfügbar
- XML Kommentare werden vom Binder nicht angetastet
- ideal zur Verarbeitung großer XML Dokumente bei denen nur kleine Teile manipuliert werden

18

Die ultimative Komponente zum manipulieren von XML Dokumenten bei denen nur ein Teil benötigt wird

Navigiert wird mit Xpath im Baum und dann der Binder an den Teilbaum gebunden

Der Binder selbst bietet Navigation im spezifizierten Teilbaum durch
`getXMLNode /getJAXBNode`

Synchronisieren der Sichten findet mit `updateXML` um das DOM zu aktualisieren bzw. `updateJAXB` um die Java Objekte zu aktualisieren

2. JAXB API

JavaBeans an DOM binden (Konzept zum Erstellen von SOAP messages)

```
...
Binder<Node> binder = context.createBinder();
//Erstellen des DOM mit dem XML file
Node mappingBodyNode =
XPathNodeLocator getNodeUsingXPath(document, xPathExpression);
JAXBElement jbelement = binder.unmarshal(mappingBodyNode,
MappingBody.class);
MappingBody mappingBody = (MappingBody)jbelement.getValue();
//Element bearbeiten
mappingBodyNode=binder.updateXML(mappingBody);
...
...
```

19

Die wichtigsten Ausschnitte sind zu sehen

Vorausgesetzt man hat eine XML Datei die ein leeres MappingBodyElement besitzt erstellt man darauf basierend ein DOM document und navigiert mittels XPath an diese Stelle der Binder gibt beim unmarshallen den leeren Mappingbody zurück (der umweg über JAXBElement ist notwendig da MappingBody kein @XMLRootElement aufweist)

Dann bearbeiten wir das Element nach Java manier und triggern die updateXML Methode des Binders



3. XML Schema to Java

Automatisierung der JavaBeans Generation

```
<echo message="Compiling the schema..." />
<xjc destdir="src" binding="src/typemapping.xjb"
removeOldOutput="yes" extension="true"
package="de.genesez.typemapping.typemappingtypes">
<produces dir="src/de/genesez/typemapping/typemappingtypes"
includes="**/*" />
<schema dir="src">
<include name="typemapping.xsd" />
</schema>
</xjc>
```

20

XJC kann via Konsole oder via ANT aufgerufen werden

Der Schemacompiler nimmt das Schema entgegen und würde damit schon zufrieden sein...

Alternativ kann man über das binding Attribut Bindungskonfigurationen angeben und mit package das Package spezifizieren weil ansonsten ein package auf Namespace Basis generiert wird.

Um sich von Altlasten zu befreien kann man alten Output entfernen lassen, wobei Änderungen an den Klassen verloren gehen (aber der Schemacompiler bietet die Möglichkeit verändertes nicht anzufassen, welche aber nicht implementiert)



3. XML Schema to Java

Bindungskonfiguration – Binding Anpassen

- Besteht aus Binding Declarations
- Anpassung des Verhaltens des SchemaCompilers
- inline oder externe Definitionen
- Aufruf via cmd shell, ant task, code

21

Bindungskonfigurationen stellen eine Art manuelle Anpassung des SchemaCompiler verhaltens dar

Sie werden i.d.r. in einem xjb file gespeichert (extern) oder inline in das Schema eingefügt

Bindungskonfigurationen bestehen aus Binding Declarations

Sie werden dem xjc als Parameter zur Verfügung gestellt



3. XML Schema to Java

Binding Declarations – Mittelsmann zwischen Schema und erzeugten JavaBean

- Eigener Namespace
- Werden über das Annotation Tag des Schemas eingebunden
- Scope:
 - Global (für alle Schemas)
 - Schema (für ein Schema)
 - Typ/Element (für global definierten Typ/Element der Schemainstanz)
 - Komponente (für Unterelement von Typ/Element)

22

<http://java.sun.com/webservices/docs/1.6/tutorial/doc/JAXBWorks6.html>



3. XML Schema to Java

Binding Declarations - Beispiel

- inline Beispiel

```
...  
<xs:element ref="tm:default">  
    <xs:annotation>  
        <xs:appinfo>  
            <jaxb:bindings>  
                <jaxb:property name="destinationMapping"  
                    generateIsSetMethod="true" />  
            </jaxb:bindings>  
        </xs:appinfo>  
    </xs:annotation>  
</xs:element>
```

23

Inline Binding declarations haben den Vorteil das sie einfach an das Element angehangen werden können

Nachteilig wirkt es sich auf die Lesbarkeit des Schemas aus

3. XML Schema to Java

Binding Declarations - Beispiel

- externe Definition

```
<jaxb:bindings node="//xs:complexType[@name='multiValuedTypeType']">
    <jaxb:class name="MultiValuedType" />
    <jaxb:bindings node=".//xs:element[@ref='tm:default']">
        <jaxb:property name="destinationMapping"
            generateIsSetMethod="true" />
    </jaxb:bindings>
</jaxb:bindings>
```

24

Vorteil liegt hier in der Extraktion des Bindings in ein xjb file um die Lesbarkeit des Schemas sowie der Binding declarations zu erhöhen

Nachteil liegt in der Navigation zu den einzelnen Elementen via XPath



3. XML Schema to Java

Elementare Binding Declarations

- <jaxb:collectionType>
 - Anpassen von Aufzählungen
- <jaxb:package>
 - Anpassen der package names
- <jaxb:class>
 - Anpassen der generierten Klassen
- <jaxb:property>
 - Anpassen der Komponenten (Member)
- <jaxb:javadoc>
 - javaDoc Kommentar anfügen

25

Die wichtigsten binding declarations um seine generierten Java Beans anzupassen



3. XML Schema to Java

Namenskonventionsproblem zwischen Java und XML

- XML kann Namenskonflikte in Java erzeugen
- Schema Compiler passt Standardmäßig Namen an
- <jaxb:nameXMLTransform> Prä-/Suffix Generierung
- enums werden mit den<jaxb:typesafeEnum*> Tags angepasst

26

XML kann typen und elemente gleich benennen und mit <jaxb:globalBindings generateElementClass=„true“> zu Problemen führen hier hilft das umbenennen der Elemente oder einsetzen von jaxb:namXMLTransform um Suffixe /Präfixe zu generieren



3. XML Schema to Java

Datentypen anpassen

- <jaxb:baseType> kann generalisieren /spezialisieren
- <jaxb:javaType> umgeht den SchemaCompiler und mappt auf die angegebene Java Klasse
- nutzt XmlAdapter Klasse zum mappen (gebunden über @XmlJavaTypeAdapter())
- Hilfsklasse DatatypeConverter übernimmt die meisten Konvertierungen
- eigene Parse und Printmethoden können über <jaxb:javaType> angegeben werden

3. XML Schema to Java

eigene Parsemethode – wenn das unmarshalling
was anderes machen soll

```
//eigene Parsemethode
public static Long parseDatetoLong(String dateString){
    Calendar c = DatatypeConverter.parseDate(dateString)
    Date d=c.getTime();
    return Long.valueOf(date.getTime());
}

//binding declaration
<jaxb:javaType name="Long"
parseMethod="mypackage.MyClass.parseDatetoLong" />
//generierter Adapter
public class Adapter1 extends XmlAdapter<String, Long>{
    public Long unmarshal(String value){
        return(mypackage.MyClass.parseDatetoLong(value));}}
```



3. XML Schema to Java

Binding Goodies

- zusätzliche Binding declarations unter
<http://java.sun.com/xml/ns/jaxb/xjc>
- <xjc:superClass>
 - globale Klasse von der alle Elemente erben
- <xjc:superInterface>
 - Root Interface
- <xjc:javaType>
 - erweiterter <jaxb:javaType> um eigene Adapter zu binden
- <xjc:simple>
 - simpler and better Binding mode

29

neben dem xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc" muss noch jaxb:extensionBindingPrefixes="xjc" angegeben werden um die goodies zu referenzieren

der xjc compiler muss zusätzlich in den extension mode geschalten werden (mit –extension)

das Root Interface am besten in Kombination mit generateValueClass="false" nutzen um Implementierungsklassen zu generieren

<xjc:simple> ist auf eigene Gefahr zu nutzen



4. Java to XML Schema

aus JavaBean ein Schema erstellen

- SchemaGeneration Einführungsbeispiel

```
//aus diesem Bean
@XmlRootElement(name="roottag")
public class MyClass{
    @XmlElement(name="subtag")
    private String subElement;
    protected String getSubElement(){return subElement;}
    protected void setSubElement(String subElement){
        this.subElement=subElement;}
//wird dieses Schema
<xs:element name="roottag" type="myClass" />
<xs:complexType name="myClass"><xs:sequence>
<xs:element name="subtag" type="xs:string minOccurs="0" />
</xs:sequence></xs:complexType>
```



4. Java to XML Schema

MAJOs erstellen

- @XmlElement: Bindet Variable an XML Element
- @XmlAttribute: Bindet Variable an Attributwert
- @XmlValue: Binding an text des XML Elements
- @XmlTransient: XML Bindung unterdrücken
- @XmlAccessorType: definiert Sortierung
- @XmlAccessorType: definiert welche Variable beim Binding beachtet werden
- @XmlElementRef: Bindet Variable an Rootelement und referenziert darauf (<xs:element ref="" />)
- @XmlRootElement: definiert die Klasse als Rootelement
- @XmlType: Binding zwischen Klasse und complexType

31

MAJO ist ein neuer Ausdruck geprägt von Java 5 Annotationen und steht für Massively annotated Java Object



4. Java to XML Schema

Java Collections und ihre Schema Pendants

- Listen / Arrays werden an Sequenz gebunden
- Komplexe Collections werden zum complexType
- Maps werden zu verschachtelten entry-key-value Elementen
- @XmlList bindet die Liste an eine <xs:list>
- @XmlElement bildet eine <xs:choice> ab
- @XmlElementWrapper schachtelt die Sequenz in ein weiteres Element
- @XmlMixed setzt mixed="true"

32

man beachte weitere Annotationen die wiederum nur für Listen gelten siehe
Glassfish Doku (@XmlList, @XmlElement, @XmlElementRefs,
@XmlElementWrapper, @XmlMixed)



4. Java to XML Schema

Java Enums sind nicht gleich XML Enum

- Enums werden standarmäßig an simpleType +
`<xs:restriction><xs:element>` gebunden
- `@XmlAttribute` passt den Basisdatentyp an
- mit `@XmlEnumValue` können Java Enum Elemente
an Xml Enums gebunden werden



4. Java to XML Schema

selfmade Typebinding um Logik an die JavaBean zu binden

- @XmlJavaTypeAdapter bindet den XmlAdapter an die Java Bean zum marshallen /unmarshallen
- implementiert beschreibt der XmlAdapter die Umwandlung von einem Java Bean sowie einem Speicherdatentyp der an XML gebunden wird
- XmlAdapter kann auch komplexe Datentypen verarbeiten



4. Java to XML Schema

Wildcards für evolutionäre Schemen

- @XmlAnyAttribute als Wildcard Attribut
- @XmlAnyElement als Wildcard Element
- @XmlAnyElement(lax=true) weißt den Unmarshaller an bekannte Elemente an ihre Java Repräsentation zu binden



4. Java to XML Schema

verschachtelte Daten für Objektgraphen

- @XmlID kann als String einer JavaKlasse als Schlüssel fungieren um auch im XML Dokument zu referenzieren (wird dann zu "xs:id")
- Kombinationen aus @XmlID und @XmlIDREF können Java Objektgraphen in XML abbilden

4. Java to XML Schema

eigene Factories – wenn die übliche Fabrik nicht das richtige herstellt

- @XmlRegistry Annotation für die Factory
- @XmlElementDecl Annotation für die FactoryMehtode

```
@XmlRegistry
public class MessageElementFactory{
    @XmlElementDecl(name = "message")
    JAXBElement<String> createMessageElement(String name){
        return new JAXBElement<String>(new
        QName("message"), String.class, name);}}
```



5. Quellen

- JAXB 2.0 Ein Programmertutorial für die Java Architecture for XML Binding
- <https://jaxb.dev.java.net/>
- <http://java.sun.com/developer/technicalArticles/WebServices/jaxb/>
- <https://jaxb-architecture-document.dev.java.net/nonav/doc/?jaxb/package-summary.html>