Westsächsische Hochschule Zwickau (FH)
University of Applied Sciences

JAXB 2.0 How to

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

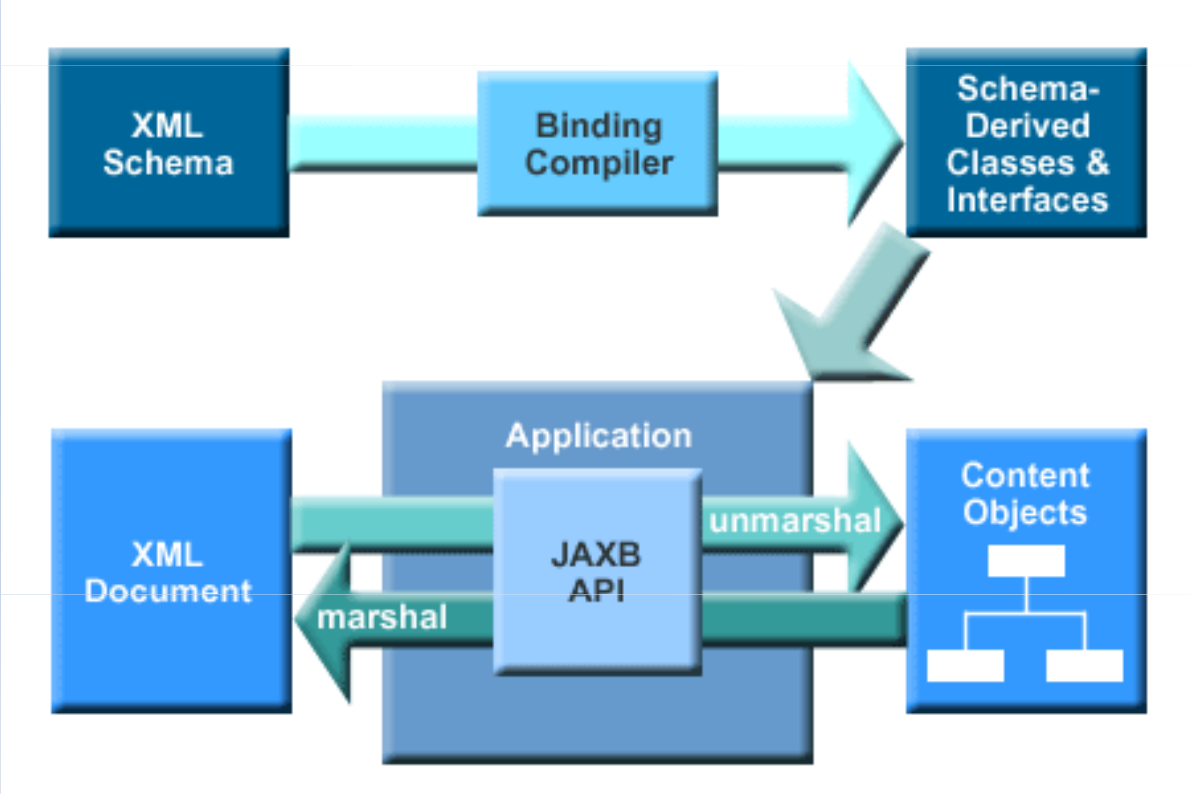
1. Requirements

Alternativen zu JAXB

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

1. Requirements

Architektur – Wie funktioniert XML Binding



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

1. Requirements

JAXB Dictionary

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

1. Requirements

Vorausgesetzte Technologien

- XML
- XML Schema
- XPath
- ANT

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;
 }
}
```

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“);
 marshaller.marshall(hello, System.out);
 }
}
```

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

2. JAXB API

Marshalling – Java zu XML

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

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
- mia JAXBElement können Teilbäume und nicht Wurzelelemente unmarshalled werden
- mittels xsi:type Element können Elemente auf anderen Elementen abgebildet werden

2. JAXB API

Non-Rootelements unmarshallen

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

2. JAXB API

Teilbäume unmarshallen

```
…
//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()
…
```

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

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

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

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.unmarshall(mappingBodyNode,
MappingBody.class);
MappingBody mappingBody = (MappingBody)jbelement.getValue();
//Element bearbeiten
mappingBodyNode=binder.updateXML(mappingBody);
…
```

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>
```

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

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)

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>
```

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>
```

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

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

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

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
- @XmlAccessorOrder: 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

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>
- @XmlElements bildet eine <xs:choice> ab
- @XmlElementWrapper schachtelt die Sequenz in ein weiteres Element
- @XmlMixed setzt mixed="true"

4. Java to XML Schema

Java Enums sind nicht gleich XML Enum

- Enums werden standarmäßig an simpleType + <xs:restriction><xs:element> gebunden
- @XmlEnum 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 Programmiertutorial 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