ARS

Member of

TIMETOACT GROUP
SOFTWARE & CONSULTING

Java API for XML Binding (JAXB)

XML Verarbeitung mit Java

ARS

Inhalte dieses Kapitels

Allgemeines

Vorgehensweisen bei der Entwicklung

Mapping-Annotationen

Verwendung im Java-Umfeld

ARS

Einführung

- High Level API
- Mapping Java-Objekte ←→ XML
  - Annotationen
  - XML-Datei ("Binding File")
- Verwendbar seit Java 5
- Teil von Java SE seit Java 6

```xml
<booking id="123798094">
  <training>JAVA-ADVANCED</training>
  <date>2008-08-24T11:32:52</date>
</booking>
```

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "booking")
public class Booking {

  @XmlAttribute(required = true)
  private long id;

  @XmlElement(required = true)
  private String training;

  @XmlElement
  @XmlSchemaType(name = "date")
  private XMLGregorianCalendar date;

  […]

}
```

ARS

Gegenüberstellung zu JAXP [1|2]

JAXB

- (XML) Binding Framework (High Level)
- Transformation XML ←→ Java-Objekte
- Vorteil gegenüber JAXP:
  - Entwicklungsaufwand beim Lesen/Schreiben von XML gering

JAXP

- (XML) Processing APIs (Low Level)
- Manuelles Parsen/Erzeugen von XML
- Vertreter: JDOM, SAX, StAX, TrAX (XSLT)
- Vorteil gegenüber JAXB:
  - Ressourcenschonende Verarbeitung (außer DOM)
  - Implementierung spezieller Szenarien

ARS

Gegenüberstellung zu JAXP [2|2]

Anwendungsgebiete von JAXB

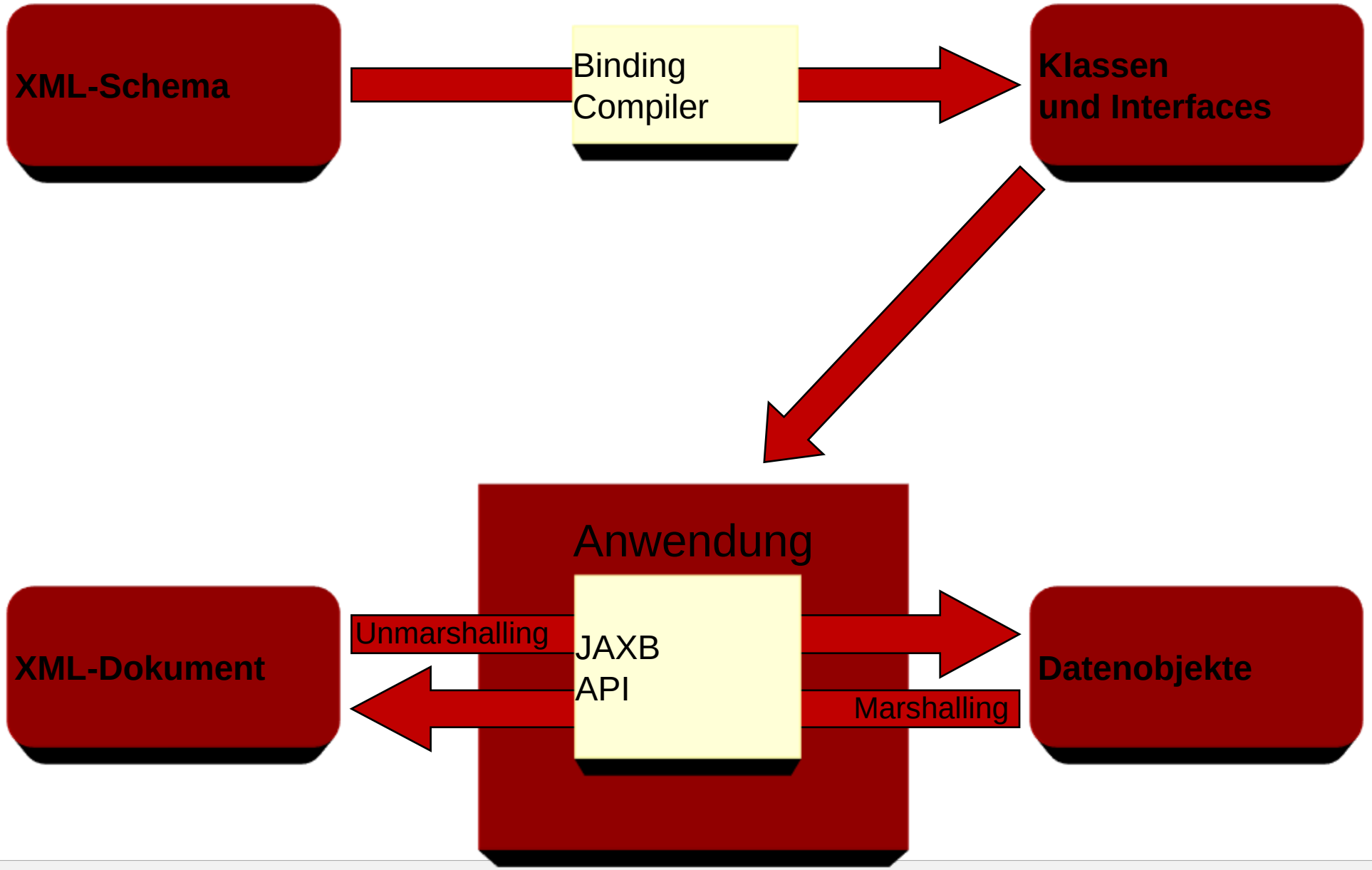
- Zugang zu Inhalten von XML-Instanzdokumenten mit gering(st)em Entwicklerаufwand (Java-Objekte statt XML-Elemente)
- Direkter, nicht-sequentieller Zugriff auf Inhalte à la DOM ohne dessen Komplexität
- Manipulieren und Schreiben von Daten im Speicher
- Vielzahl von XML-Instanzdokumenten

Voraussetzungen für JAXB

- XML-Dokumente mit geringer Größe

ARS

Vorgehensweisen bei der Entwicklung [1|4]

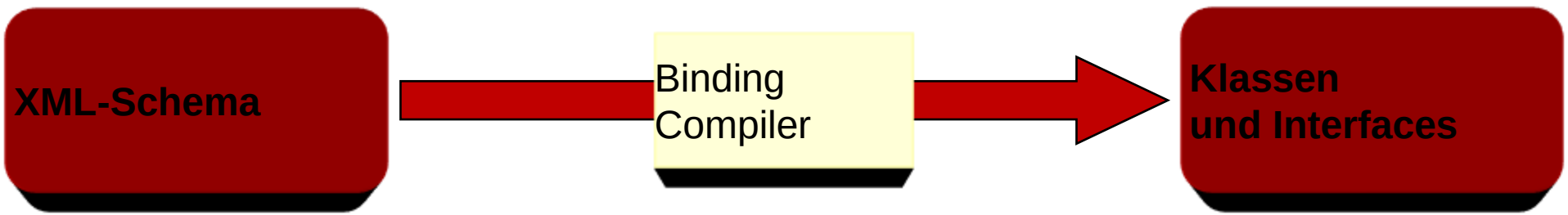


ARS

Vorgehensweisen bei der Entwicklung [2|4]

Binding Compiler

- Kommandozeilentool `xjc`
- Code-Generierung auf Basis eines XML-Schemas
- Top-Down-Ansatz
- Konfiguration optional über Binding File (`*.xjb`)
  - Java-Package
  - Java-Datentypen (z.B. Enumerations)
  - Eigenschaften der generierten Klassen (z.B. `Serializable`)



ARS

Vorgehensweisen bei der Entwicklung [3|4]

Binding File (Beispiel)

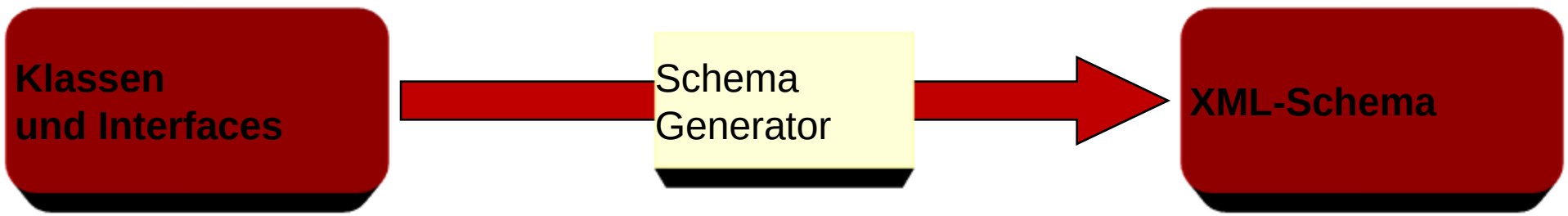
```
<bindings version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://java.sun.com/xml/ns/jaxb" xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
  extensionBindingPrefixes="xjc">

 <globalBindings>
  <xjc:serializable uid="1" />
 </globalBindings>

 <bindings schemaLocation="../src/main/resources/META-INF/schemas/daojones-2.0-beans.xsd">
  <bindings
    node="//xs:complexType[@name='DatabaseTypeMapping']/xs:attribute[@name='type']/xs:simpleType">
   <typesafeEnumClass name="DataSourceType" />
  </bindings>
 </bindings>


</bindings>
```

ARS

Vorgehensweisen bei der Entwicklung [4|4]

Vom Code zum Schema

- Bottom-Up-Ansatz
- Generierung eines XML-Schemas durch Analyse von Java-Klassen
- Kommandozeilentool `schemagen`
  - Konfiguration optional über
    - Binding File
    - Annotationen

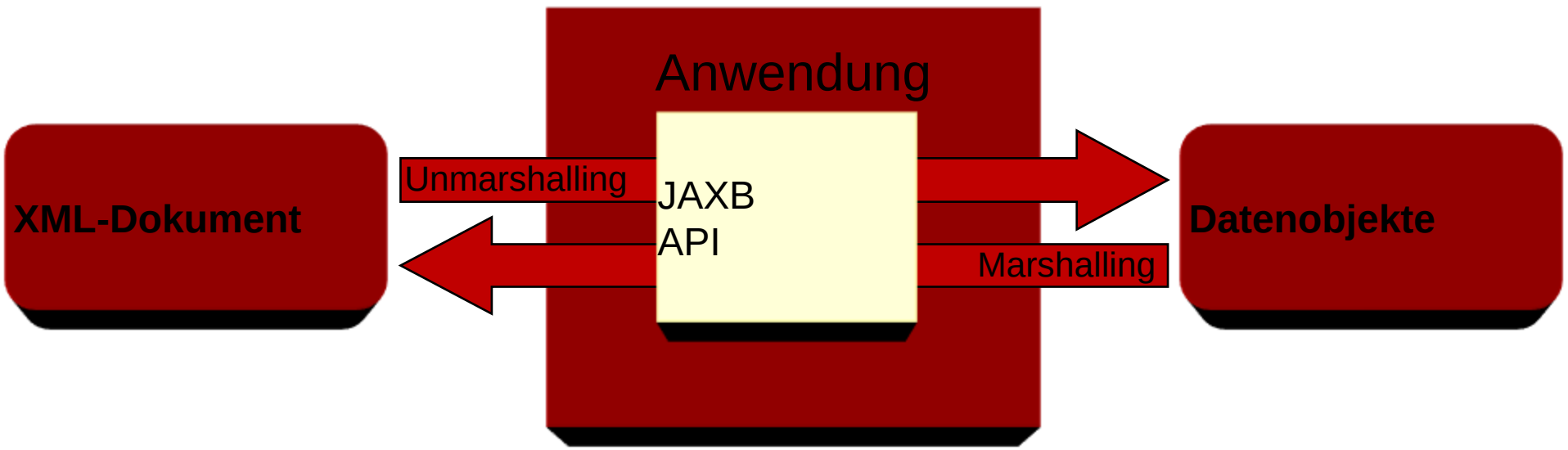


ARS

Parsen und Serialisieren: Unmarshalling und Marshalling

Beispiel

```
File file = new File("booking.xml");
Booking booking= new Booking();
// Marshalling Object -> XML
JAXB.marshal(booking, file);
// Unmarshalling XML -> Object
Booking booking2 = JAXB.unmarshal(file, Booking.class);
```



ARS

Mapping-Annotationen [1|7]

- `@XmlAccessorType`
  - Konfiguration JAXB Processor
  - *NONE* / *FIELD* / *PROPERTY* / *PUBLIC_MEMBER* / *NONE*
  - Default: *FIELD*
- `@XmlRootElement`
  - Name des Wurzelelements
  - Ggf. Namespace
- `@XmlType`
  - Alternativ (oder zusätzlich) zu `@XmlRootElement`
  - Name / Namespace / Reihenfolge der Properties
- ...

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "booking")
@XmlType(propOrder = { "training", "date" })
public class Booking {
  private long id;
  private String training;
  private XMLGregorianCalendar date;
}
```

ARS

Mapping-Annotationen [2|7]

- `@XmlAttribute`
  - Definition von Attributen des Wurzelelements
  - Attribute:
    `name`, `namespace`, `required`
- `@XmlElement`
  - Konfiguration eines XML Elements
  - Attribute:
    `required`, `defaultValue`, `nillable`, `name`, `namespace`, `type`

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "booking")
@XmlType(propOrder = { "training", "date" })
public class Booking {
  @XmlAttribute(required = true)
  private long id;
  @XmlElement(required = true)
  private String training;
  @XmlElement
  @XmlSchemaType(name = "date")
  private XMLGregorianCalendar date;
}
```

ARS

Mapping-Annotationen [3|7]

- `@XmlList`
  - Für Listen aus „SimpleTypes“
- `@XmlElementWrapper`
  - Für Listen aus „ComplexTypes“
- `@XmlValue`
  - Feld als Wert des zur Klasse gehörigen Wurzelelements
- `@XmlTransient`
  - Feld wird ingoriert
- … viele weitere für Vererbungsbeziehungen, Mapping von JavaTypen zu XML Typen, usw.

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "booking")
@XmlType(propOrder = { "training", "date" })
public class Booking {
  @XmlAttribute(required = true)
  private long id;
  @XmlElement(required = true)
  private String training;
  @XmlElement
  @XmlSchemaType(name = "date")
  private XMLGregorianCalendar date;
}
```

ARS

Mapping-Annotationen [4|7]

Annotation der Modellklassen

```java
@XmlRootElement(name = "auto")
@XmlAccessorType(XmlAccessType.NONE)
public class Auto {

  @XmlAttribute
  private Integer id;

  @XmlElement
  private String hersteller;

  @XmlElement
  private String modell;

  @XmlElement
  private String baujahr;

}
```

```xml
<?xml version="1.0" encoding="UTF-8">
<auto id="1">
    <hersteller>BMW</hersteller>
    <modell>Z8</modell>
    <baujahr>2013</baujahr>
</auto>
```

ARS

Mapping-Annotationen [5|7]

Binding für Listen: Standard = Problem

```
@XmlRootElement(name = "garage")
@XmlAccessorType(XmlAccessType.FIELD)
public class Garage {

  @XmlElement
  private List<Auto> autos;

}
```

```
<?xml version="1.0" encoding="UTF-8">
<garage>
    <autos id="1">
        <hersteller>BMW</hersteller>
        <modell>Z8</modell>
        <baujahr>2013</baujahr>
    </autos>
    <autos id="2">
        <hersteller>BMW</hersteller>
        <modell>Z3</modell>
        <baujahr>2009</baujahr>
    </autos>
</garage>
```

ARS

Mapping-Annotationen [6|7]

Binding für Listen: Lösung 1

```java
@XmlRootElement(name = "garage")
@XmlAccessorType(XmlAccessType.FIELD)
public class Garage {

  @XmlElement(name="auto")
  private List<Auto> autos;

}
```

```xml
<?xml version="1.0" encoding="UTF-8">
<garage>
    <auto id="1">
        <hersteller>BMW</hersteller>
        <modell>Z8</modell>
        <baujahr>2013</baujahr>
    </auto>
    <auto id="2">
        <hersteller>BMW</hersteller>
        <modell>Z3</modell>
        <baujahr>2009</baujahr>
    </auto>
</garage>
```

ARS

Mapping-Annotationen [7|7]

Binding für Listen: Lösung 2

```java
@XmlRootElement(name = "garage")
@XmlAccessorType(XmlAccessType.FIELD)
public class Garage {

  @XmlElementWrapper(name="autos")
  @XmlElement(name="auto")
  private List<Auto> autos;

}
```

```xml
<?xml version="1.0" encoding="UTF-8">
<garage>
    <autos>
        <auto id="1">
            <hersteller>BMW</hersteller>
            <modell>Z8</modell>
            <baujahr>2013</baujahr>
        </auto>
        <auto id="2">
            <hersteller>BMW</hersteller>
            <modell>Z3</modell>
            <baujahr>2009</baujahr>
        </auto>
    </autos>
</garage>
```

ARS

Java-Umfeld

Webservices mit JAX-WS

- Nachrichten zwischen Webservices in XML-Format (SOAP)
  - Marshalling und Unmarshalling durch Laufzeitumgebung
  - Einfluss auf Serialisierung eigener Datentypen per JAXB durch
    - Annotationen
    - Binding File
- Ansätze analog JAXB
  - Top-Down: Vom WSDL zum Java-Code
  - Bottom-Up: Vom Java-Code zum WSDL
- Webservice-Clients
  - Dynamic Proxy Client (Standard) nutzt JAXB
  - Dispatch Client nutzt JAXP