

Web Services

Reinhold Rumberger
rumberd@studi.informatik.uni-stuttgart.de

Inhalt

- Theoretischer Teil
 - SOA
 - Web Services Grundlagen
 - SOAP
 - WSDL
 - UDDI
 - Java und Web Services
 - JAX-WS 2.0 (JSR-224)
 - WS-Metadata 2.0 (JSR-181)
 - JAXB 2.0 (JSR-222)
- Praktischer Teil

Theoretischer Teil

SOA

- Architekturstil
- Lose gekoppelte Dienste
 - Kommunikation
 - Flexibilität
- SOA definiert Kommunikationsmethoden

⇒ SOA kann durch Web Services implementiert werden.

Web Services Grundlagen

- Plattformunabhängig
- RESTful WS
 - Nicht wohldefiniert
 - Verwendet HTTP direkt
 - Kennt nur Operationen wie GET, PUT, DELETE
⇒ für uns eher uninteressant
- Message-oriented WS
 - Auch bekannt als „Big Web Services“
 - Verwenden XML + SOAP (+ WSDL)
 - Kontraktbasierte Kommunikation
⇒ Lose Kopplung

Web Services Grundlagen

■ WS-I

- Profiles: Bestimmte Spezifikationen & Versionen + zusätzliche Einschränkungen
 - WS-I Basic Profile
(SOAP 1.1, HTTP/1.1, WS-Addressing 1.0 (Core, SOAP-Binding, WSDL-Binding), XML-Binary Optimized Packaging...)

■ Zusätzliche WS-Spezifikationen

- WS-Security
- WS-Reliability
- WS-Metadata

SOAP

- Protokoll zum Austausch strukturierter Daten
- XML-basiert
 - Leichter lesbar für Menschen
 - Hoher Aufwand beim Parsen
- Andere Protokolle zum Transport (HTTP, SMTP, FTP, ...)
 - Manche Protokolle werden von den meisten Firewalls geblockt
- Basis der Kommunikation zwischen Web Services
- Beispiel:

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
  </s:Header>
  <s:Body>
  </s:Body>
</s:Envelope>
```

WSDL

- Beschreibt die öffentliche Schnittstelle von Web Services
- Basiert auf XML

WSDL - Aufbau

- Root-Element
 - Definitions
- Imports
- Dokumentation
 - documentation

- Zugriffsdetails
 - binding
 - operation
 - input
 - output
- Endpoints
 - service
 - port

- Abstraktes Interface
 - types
 - XML-Schema
 - portType
 - operation
 - input
 - output
 - message
 - Part

UDDI

- Service zum Auffinden von Web Services
- Hält WSDL & Metadaten bereit
- Mittlerweile nur noch unternehmensintern
- Jedes Unternehmen hat eigene Standards

- Hat in der Industrie kaum noch Relevanz

Java und Web Services

- **Handlers**
 - Modifizieren Nachrichten vor und nach Bearbeitung durch Web Service
- **Service Endpoint Interface (SEI)**
 - Ein Java-Interface das Definitionen zum abstrakten WSDL-Interface enthält
- **Java-spezifische Standards**
 - JAX-WS (JSR-224)
 - WS-Metadata (JSR-181)
 - JAXB (JSR-222)

JAX-WS 2.0 (JSR-224)

- Nachfolger von JAX-RPC (JSR-101)
 - JAX-RPC war für RPC-basierte Web Services ausgelegt
- JAX-WS definiert:
 - Standard WSDL 1.1 \Leftrightarrow Java Mappings
 - Standard SOAP-Binding
 - Standard HTTP-Binding
 - Standard Handler Framework
 - Client- und Server-APIs
 - ...

WS-Metadata 2.0 (JSR-181)

- Einfacheres WS-Entwicklungsmodell
 - Ersetzt die meisten *deployment descriptors*
 - Unabhängig von der Laufzeitumgebung
- Nicht jeder WS kann implementiert werden
- Auf den folgenden Folien:
 - Die wichtigsten Annotationen
 - Deren wichtigste Attribute
 - Gekürzte Beschreibungen der Parameter

@WebService

- Markiert Web Services, die SEIs implementieren
- Attribute
 - *name*: Name im `wsdl:portType`
 - *serviceName*: Name im `wsdl:service`
 - *portName*: Name im `wsdl:port`
 - *targetNamespace*: Gibt den targetNamespace im WSDL-Dokument an
 - *wsdlLocation*: vordefiniertes WSDL-Dokument
 - *endpointInterface*: spezifiziert das implementierte SEI
⇒ Wird vom JAX-WS Runtime Environment geprüft
- Anwendbar auf Klassen und Interfaces

```
@WebService
public class Examples implements ServiceEndpointInterface {
}
```

@WebMethod

- Attribute
 - *operationName*: Name im `wsdl:operation`
 - *exclude*: Gibt an, dass die Methode nicht veröffentlicht werden soll
- Anwendbar auf Methoden

```
@WebMethod  
public void exampleMethod(){  
  
}
```

@Oneway

- Anwendbar auf Methoden
 - Kein Return-Wert
 - Keine definierten Exceptions
 - Keine OUT- bzw. INOUT-Parameter
-
- Fehlermeldung, falls diese Bedingungen nicht eingehalten werden

```
@webMethod  
@Oneway  
public void exampleMethod(){  
  
}
```


@WebParam

- Gibt an, wie ein Parameter im WSDL-Dokument aussieht
- Attribute
 - *name*: Name des Parameters im WSDL-Dokument
 - *partName*: Name im `wsdl:part`, bei RPC-Stil
 - *targetNamespace*: Namespace des Parameters
 - *mode*: IN, OUT oder INOUT
 - *header*: Ob der Parameter im Nachrichtenkopf abgelegt ist
- Anwendbar auf Parameter

```
@WebMethod  
public void exampleMethod(@WebParam int i){  
  
}
```

@WebResult

- Gibt an, wie ein Rückgabewert im WSDL-Dokument aussieht
- Attribute
 - *name*: Name der Ausgabe im WSDL-Dokument
 - *partName*: Name im `wsdl:part`, bei RPC-Stil
 - *targetNamespace*: Namespace der Ausgabe
 - *header*: Ob die Ausgabe im Nachrichtenkopf abgelegt ist
- Anwendbar auf Methoden

```
@WebMethod  
@WebResult  
public void exampleMethod(){  
  
}
```

@HandlerChain

- Erlaubt es, eine Folge von Handlern für Web Services anzugeben
- Attribute
 - *File*: Referenziert eine Datei, die eine HandlerChain definiert
- Anwendbar auf Klassen, Methoden und Felder

```
@WebService  
@HandlerChain(file="handlerChain.xml")  
public class Examples implements ServiceEndpointInterface {  
  
}
```

@SOAPBinding

- Modifiziert das SOAP-Binding
- Default: Document – Literal – Wrapped
- Attribute
 - *Style*: Kodierungsstil; DOCUMENT oder RPC
 - *Use*: Formatierungsstil; LITERAL oder ENCODED
 - *parameterStyle*: Ob der Nachrichtenkörper nur die Parameter enthält (BARE) oder ein Element, das nach der Methode benannt ist und die Parameter enthält (WRAPPED)
- Anwendbar auf Klassen und Methoden

```
@WebService
@SOAPBinding
public class Examples implements ServiceEndpointInterface {

}
```

JAXB 2.0 (JSR-222)

- Binding von Java-Klassen an XML-Schemas
 - Java-Typen \Leftrightarrow XML-Elemente
 - *Marshalling*: XML \rightarrow Java
 - *Unmarshalling*: Java \rightarrow XML
 - Generiertes XML kann validiert werden (\rightarrow JAXP 1.3)
- In JEE 5 enthalten
- Package: `javax.xml.bind`
- Wird benötigt wenn ein WS Typen benutzt, für die kein (sinnvolles) Standard-Binding existiert.

@WebServiceProvider

- Markiert Web Services die `javax.xml.ws.Provider` implementieren
- Direkter Zugriff auf XML-Nachricht
 - ⇒ Weniger Overhead (un-/marshalling entfällt)
- Für weniger komplizierte WS ungeeignet

```
@WebServiceProvider
public class Examples implements Provider<SAXSource> {

    @Override
    public SAXSource invoke(SAXSource request) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

Praktischer Teil

End Of Document