Web Services
JAX-WS
(Java API for XML-Based Web Services)

Prof. Dr. Ch. Reich
rch@fh-furtwangen.de
http://www.informatik.fh-furtwangen.de/~reich/

Web Service APIs

- Java API for XML-Based Web Services (JAX-WS)
  - fundamental technology for developing SOAP based Java Web services (JSR 224).
- Java API for XML-Based RPC (JAX-RPC)
  - uses SOAP 1.1 (JSR 101). use JAX-WS
- Java Archit. for XML Binding (JAXB; JSR 222).
- SOAP with Attachments API for Java (SAAJ; JSR 67).
- Java API for XML Processing (JAXP; JSR 206).

Frameworks supporting JAX-WS

- Apache JCX (http://incubator.apache.org/cxf/)
  - Support for Standards
    - JAX-WS, JAX-WSA, JSR-181, and SAAJ
    - SOAP 1.1, 1.2, WS-I BasicProfile, WS-Security, WS-Addressing, WS-RM and WS-Policy
    - WSDL 1.1 and 2.0
    - MTOM
- JAVA EE Application Server (Bea, Geronimo,...)

JAX-WS Web Service Step by Step

1. Create a Service Endpoint Interface (SEI) that defines the methods you wish to expose as a service.
2. Add the required annotations to your code.
3. Generate the WSDL contract for your service.
4. Publish the service.

1. Create a Service Endpoint Interface (SEI)

- SEI is the piece of Java code that is shared between a service and the consumers.
- When starting with a WSDL contract, the SEI is generated by the code generators. However, when starting from Java, it is the up to a developer to create the SEI.
- 2 ways for SEI development:
  - nothing existing: write java interface
  - existing implementation: write fitting interface

ICalculator Interface

```
public interface ICalculator
{
  public int computeSumOf(int a, int b);
}
```

Calculator Implementation

```
public class Calculator
        implements ICalculator {
   public int computeSumOf(int a, int b) {
        return a + b;
   }
}
```

2. Add the required annotations to your code.

```
import javax.jws.*;

@WebService(name="Calculator",
   targetNamespace="http://cxf.apache.org",
   serviceName="calculatorService",
wsdlLocation="http://localhost:8080/calcula
torService?wsdl",
   portName="calculatorPort")

public interface ICalculator
{
  public int computeSumOf(int a, int b);
}
```

Annotations

- `name="Calculator":` the value of the name attribute of the wsdl:portType element.
- `targetNamespace="http://cxf.apache.org":` the target namespace of the service
- `serviceName="calculatorService":` the value of the name of the wsdl:service element.
- `wsdlLocation="http:...?wsdl":` where the service will publish its WSDL contract.
- `portName="calculatorPort":` the value of the name attribute of the wsdl:port element.

Calculator Implementation

```
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService(endpointInterface="ICalculator")
@SOAPBinding(style=Style.RPC)
public class Calculator
        implements ICalculator {
  public int computeSumOf(int a, int b) {
        return a + b;
  }
}
```

Annotations

- `@WebService(endpointInterface="ICalculator")`: defines the endpoint interface.
- `@SOAPBinding(style=Style.RPC)`: each message part within the SOAP body is a parameter or return value and will appear inside a wrapper element within the soap:body element.

Client Code (1)

```
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Endpoint;
import javax.xml.ws.Service;
public class SimpleWebServiceExample {
 public static void main(String[] args)
                                  throws Exception {
  Endpoint endpoint = null;
  try {
   // publish service
   endpoint = Endpoint.publish(
     "http://localhost:8080/Calculator",
     new Calculator());
```

Client Code (2)

```
  // create service
  Service service = Service.create(new URL(
   "http://localhost:8080/Calculator?wsdl"),
   new QName("http://tutorials.de/",
   "Calculator"));
  // use service
  ICalculator simpleService =
         service.getPort(ICalculator.class);
  System.out.println(
    simpleService.computeSumOf(11, 12));
   } finally { endpoint.stop();
  }
 }
}
```

JAX-WS Annotations (1)

- JAX-WS uses annotations (relies on Java 5).
- JAX-WS annotations specify the meta data often specified in xml-files.
- JAX-WS annotations can secify:
  - target namespace
  - name of the class of the web service
  - name of the class that hold the response message
  - ...

JAX-WS Annotations (2)

- If an annotation is missing => most of it have defaults and do not need to be specified.
- Attention: Defaults with new framework versions could change.
- Better: Provide the information through the annotations, then it might work in the future as well.
- Required Annotation: `@WebService`

@Webservice Annotation Properties (1)

- name:
  - name of the service interface. This property is mapped to the name attribute of the wsdl:portType element that defines the service's interface in a WSDL contract. The default is to append PortType to the name of the implementation class.
- targetNamespace:
  - target namespace under which the service is defined. If this property is not specified, the target namespace is derived from the package name.

@Webservice Annotation Properties (2)

- serviceName
  - Specifies the name of the published service. This property is mapped to the name attribute of the wsdl:service element that defines the published service. The default is to use the name of the service's implementation class.
- wsdlLocation
  - Specifies the URI at which the service's WSDL contract is stored. The default is the URI at which the service is deployed.

@Webservice Annotation Properties (3)

- endpointInterface
  - Specifies the full name of the SEI that the implementation class implements. This property is only used when the attribute is used on a service implementation class.
- portName
  - Specifies the name of the endpoint at which the service is published. This property is mapped to the name attribute of the wsdl:port element that specifies the endpoint details for a published service. The default is the append Port to the name of the service's implementation class

Optional Annotations

- `@SOAPBinding` annotation defined by the `javax.jws.soap.SOAPBinding` interface.
  - Provides details about the SOAP binding.
  - `@SOAPBinding` annotation is not specified, a service is published using a wrapped doc/literal SOAP binding.
  - `@SOAPBinding` annotation can be put on any method.

The `@SOAPBinding` annotation

- style (Style.DOCUMENT (default); Style.RPC)
  - DOCUMENT: valid XML document
  - RPC: Each message part within the SOAP body has a parameter or return value and will appear inside a wrapper element within the soap:body element.
- use (Use.LITERAL (default); Use.ENCODED)
  - Specifies how the data of the SOAP message is streamed.

The `@SOAPBinding` annotation

- parameterStyle (ParameterStyle.BARE; ParameterStyle.WRAPPED (default))
  - Specifies how the method parameters, which correspond to message parts in a WSDL contract, are placed into the SOAP message body.

@WebMethod annotation

- @WebMethod annotation is defined by the javax.jws.WebMethod interface.
- It is placed on the methods
- @WebMethod provides the information that is represented in the wsdl:operation element.

@WebMethod annotation

- Property values:
  - operationName: value of the wsdl:operation element's name. Default: method name
  - action: value of the soapAction attribute of the soap:operation element generated for the method. The default value is an empty string.
  - exclude: Specifies if the method should be excluded from the service interface. The default is false.

`@RequestWrapper` annotation

- `@RequestWrapper` is defined by the `javax.xml.ws.RequestWrapper` interface.
- Placed on the method.
- `@RequestWrapper` specifies the Java class that implements the wrapper bean for the method parameters that are included in the request message sent in a remote invocation.
- Also used to specify the element names, and namespaces, used by the runtime when marshalling and unmarshalling the request messages

`@RequestWrapper` annotation

- Property values:
  - localName: name of the wrapper element in the XML representation of the request message. Default: value is the name of the method or the value of the @WebMethod annotation's operationName property.
- targetNamespace: namespace under which the XML wrapper element is defined. Default: value is the target namespace.
- className: Specifies the full name of the Java class that implements the wrapper element.

`@WebFault` annotation

- `@WebFault` annotation is defined by the `javax.xml.ws.WebFault` interface.
- It is placed on methods that throw exceptions.
- `@WebFault` annotation is used to map the Java exception to a `wsdl:fault` element. Information is used to marshall the exceptions into a representation that can be processed by both the service and its consumers.

`@WebFault` annotation

- Property values:
  - name: local name of the fault element.
  - targetNamespace: namespace under which the fault element is defined. Default: target namespace.
  - faultName: full name of the Java class that implements the exception.

`@OneWay` annotation

- `@OneWay` annotation is defined by the `javax.jws.OneWay` interface.
- It is placed on the methods that will not require a response from the service.
- `@OneWay` annotation tells the run time that it can optimize the execution of the method by not waiting for a response and not reserving any resources to process a response.

Defining Parameters with Annotations

The method parameters in the SEI correspond to the `wsdl:message` elements and their `wsdl:part` elements.

JAX-WS provides annotations that allow you to describe the `wsdl:part` elements that are generated for the method parameters.

- **The `@WebParam` annotation**
- **The `@WebResult` annotation**

`@WebParam` annotation

- `@WebParam` annotation is defined by the `javax.jws.WebParam` interface.
- It is placed on the parameters on the methods defined in the SEI.
- The `@WebParam` annotation allows you to specify the direction of the parameter, if the parameter will be placed in the SOAP header, and other properties of the generated `wsdl:part`.

`@WebParam` annotation

- Property values:
  - name: name of the parameter in the WSDL.
  - targetNamespace: namespace for the parameter.
  - mode (Mode.IN (default); Mode.OUT; Mode.INOUT): direction of the parameter.
  - header (false (default); true): parameter is passed as part of the SOAP header.
  - partName: value of the name attribute of the wsdl:part element for the parameter when the binding is document.

`@WebResult` annotation

- `@WebResult` annotation is defined by the `javax.jws.WebResult` interface.
- It is placed on the methods defined in the SEI.
- The `@WebResult` annotation allows you to specify the properties of the generated `wsdl:part` that is generated for the method's return value

`@WebResult` annotation

- Property values:
  - name: name of the return value in the WSDL. Default: return.
  - targetNamespace: namespace for the return value.
  - header: return val. passed as part of SOAPheader.
  - partName: value of the name attribute of the wsdl:part element for the return value

Example with lot of annotations

```
import javax.jws.*;
import javax.xml.ws.*;
import javax.jws.soap.*;
import javax.jws.soap.SOAPBinding.*;
Import javax.jws.WebParam.*;
@WebService(name="quoteReporter")
@SOAPBinding(style=Style.RPC, use=Use.LITERAL)public interface
   quoteReporter
{  @WebMethod(operationName="getStockQuote")
   @RequestWrapper(targetNamespace="http://demo.iona.com/types",
         className="java.lang.String")
   @ResponseWrapper(targetNamespace="http://demo.iona.com/types",
               className="org.eric.demo.Quote")
   @WebResult(targetNamespace="http://demo.iona.com/types",
   name="updatedQuote")  public Quote getQuote(
         @WebParam(targetNamespace="http://demo.iona.com/types",
                     name="stockTicker", mode=Mode.IN)
                     String ticker  );}
```