JAX-WS – Creating Java-based SOAP and WS-* Services

Oxford University

Software Engineering Programme

Sep 2015

Contents

- Why JAX-WS
- Origins and history
- Introduction
- Examples
- Further resources

JAX-WS Motivation

- Java API for XML Web Services
  - Currently version 2.2
- Create a standard Java approach to creating and consuming SOAP/WSDL web services
- Based on annotations
- Work with WS-I Basic Profile
- Work with JAX-B (Java API for XML Binding)
- Replaced the (broken) JAX-RPC specification

Two approaches

- Code first:
  - Create Java code, annotate
  - Run Java2WS to create WSDL / XSD etc
- Contract first:
  - Create (or re-use) WSDL / XSD etc
  - run WSDL2Java to create the Java artefacts

Tool names and syntax ARE NOT part of the spec

- In theory, tools should work with other spec implementations
  - Since the created artefacts should be portable
- In practise, I've never tested this
  - Java2WS is equivalent to wsgen
  - WSDL2Java is equivalent to wsimport

Code first (annotated POJOs)

- Start with a **P**lain **O**ld **J**ava **O**bject
- Create annotations that document the service definition, binding approach, etc

Common Annotations

- @WebService
- @SOAPBinding
- @WebMethod
- @WebParam
- @OneWay
- @HandlerChain

WebService

Applies to class or interface

All parameters are optional

```
@WebService
    (name = “OrderService”,
    serviceName = "OrderProcess",
     portName = "OrderProcessPort",
    targetNamespace = "http://freo.me/order",
    wsdlLocation=“path to existing wsdl”)
```

WebService continued

@WebService(endpointInterface = “me.freo.OrderProcess") applies to class only

This allows you to create an interface defining the service/WSDL and a separate implementation. This is especially important for WSDL first operation

SOAPBinding

Applies to class or interface

```
@SOAPBinding(
  style=SOAPBinding.Style.DOCUMENT,
  use=SOAPBinding.Use.LITERAL,
     parameterStyle=
     SOAPBinding.ParameterStyle.WRAPPED)
```

My hint: ALWAYS use Doc/Lit/Wrapped

see http://pzf.fremantle.org/2007/05/handlign.html

Second hint: this is the default so don't use @SOAPBinding!

WebMethod

Applies to Method

```
@WebMethod(
    action="MySOAPAction", //optional
    operationName="myWSDLop",
    exclude=true) // do NOT expose this
                  //  inherited method
```

OneWay

- Applies to a method that is marked @WebMethod
- Indicates that there is no response expected
- Assuming this is over HTTP, there should just be a HTTP 202 Accepted response
- Over JMS, no response message expected

WebParam

- A way of defining the mapping between the XML/SOAP message and the Java Parameters

```
@WebParam(
        name="nameOfXMLElement",
        partName="nameOfWSDLPart",
        targetNamespace="xmlNamespace",
        mode="IN|OUT|INOUT",
        header=true|false)
```

Java2WS tooling

java2ws -databinding <jaxb or aegis> -frontend <jaxws or simple>
-wsdl -wrapperbean -client -server -ant -o <output-file>
-d <resource-directory> -classdir <compile-classes-directory>
-cp <class-path> -soap12 -t <target-namespace>
-beans <ppathname of the bean definition file>*
-address <port-address> -servicename <service-name>
-portname <port-name> -createxsdimports -h -v -verbose
-quiet {classname}

Options

Option Interpretation

-?,-h,-help Displays the online help for this utility and exits.

-o Specifies the name of the generated WSDL file.

--databinding Specify the data binding (aegis or jaxb). Default is jaxb for jaxws frontend, and aegis for simple frontend.

-frontend Specify the frontend to use. jaxws and the simple frontend are supported.

Options (continued)

-wsdl Specify to generate the WSDL file.

-wrapperbean Specify to generate the wrapper and fault bean

-client Specify to generate client side code

-server Specify to generate server side code

-ant Specify to generate an Ant build.xml script

-cp Specify the SEI and types class search path of directories and zip/jar files.

-soap12 Specifies that the generated WSDL is to include a SOAP 1.2 binding.

-t Specifies the target namespace to use in the generated WSDL file.

-servicename Specifies the value of the generated service element's name attribute.

-v Displays the version number for the tool.

Options (continued)

-verbose Displays comments during the code generation process.

-quiet Suppresses comments during the code generation process.

-s The directory in which the generated source files(wrapper bean ,fault bean ,client side or server side code) are placed.

-classdir The directory in which the generated sources are compiled into. If not specified, the files are not compiled.

Options (continued)

-portname Specify the port name to use in the generated wsdl.

-address Specify the port address.

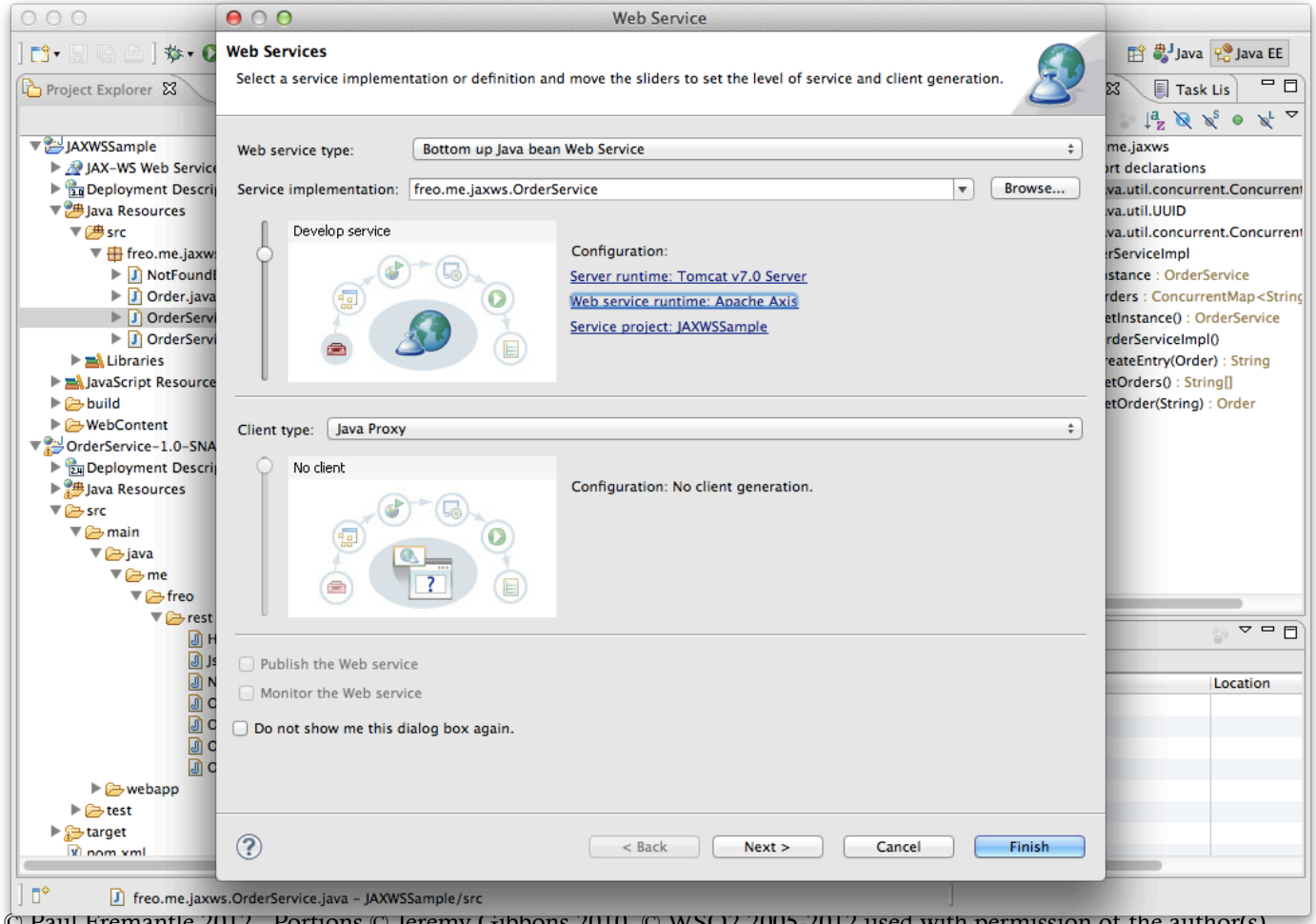
-beans Specify the pathname of a file defining additional Spring beans to customize databinding configuration.

-createxsdimportsOutput schemas to separate files and use imports to load them instead of inlining them into the wsdl.

-d The directory in which the resource files are placed, wsdl file will be placed into this directory by default

classname Specifies the name of the SEI class.

But you can ignore that!



HandlerChain

@HandlerChain(file = "handlers.xml")

Handlers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-name>Transform</handler-name>
      <handler-class>org.freo.jaxws.handlers.Transform</handler-class>
    </handler>
    <handler>
      <handler-name>Log</handler-name>
      <handler-class>org.freo.jaxws.handlers.Logger</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

WSDL first

- Again there is a tool for this
- You might want to create a service
  - Contract-first (design the WSDL, then implement)
  - Implement a standard WSDL
  - Re-architect an existing service
  - Copy a competitor's service (though this is a thorny issue!)
- Very likely you need to call a service

WSDL2Java

wsdl2java -fe|-frontend <front-end-name> -db|-databinding <data-binding-name>
-wv <wsdl-version> -p <[wsdl-namespace =]package-name>* -sn <service-name>
-b <binding-file-name>* -reserveClass <class-name>* -catalog <catalog-file-name>
-d <output-directory> -compile -classdir <compile-classes-directory> -impl -server
-client -all -autoNameResolution -allowElementReferences|-aer<=true>
-defaultValues<=class-name-for-DefaultValueProvider> -ant
-nexclude <schema-namespace [= java-package-name]>* -exsh <(true, false)> -noTypes
-dns <(true, false> -dex <(true, false)> -validate -keep
-wsdlLocation <wsdlLocation> -xjc<xjc-arguments>* -asyncMethods<[=method1,method2,...]>*
-bareMethods<[=method1,method2,...]>* -mimeMethods<[=method1,method2,...]>* -noAddressBinding
-faultSerialVersionUID <fault-serialVersionUID> -exceptionSuper <exceptionSuper>
-mark-generated -h|-?|-help -version|-v -verbose|-V -quiet|-q|-Q -wsdlList <wsdlurl>

Again do this via the Eclipse tooling!

Resources

- The Labs
- The Spec
  - http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index3.html
- The CXF documentation
  - http://cxf.apache.org/docs/a-simple-jax-ws-service.html
- The Reference Implementation
  - http://jax-ws.java.net/