JAX-WS Basics

Sang Shin
Michèle Garoche
www.javapassion.com
"Learning is fun!"

Agenda

- Quick overview of JAX-WS
- JAX-WS programming Model
  - Layered programming model
  - Server side
  - Client side
- Review of annotations
- Protocol and transport independence
- Client catalog file

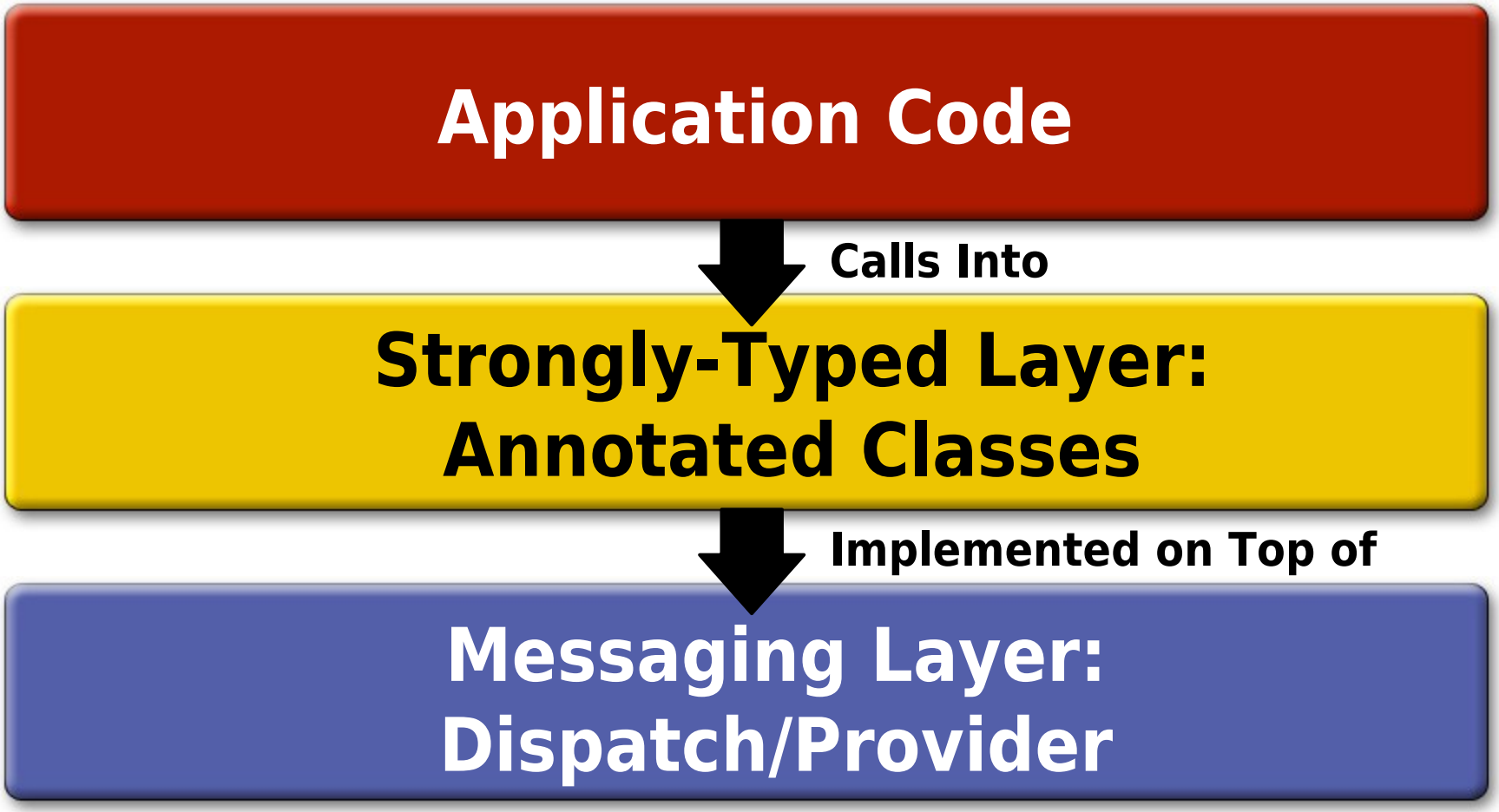
Quick Overview of JAX-WS 2.0

Quick Overview of JAX-WS 2.0

- Simpler way to develop/deploy Web services
  - Plain Old Java Object (POJO) can be easily exposed as a Web service through annotation
  - No deployment descriptor is needed
  - Layered programming model
- Part of Java SE 6 version 4 and after and Java EE 5 platforms
- Built-in data binding via JAXB 2.0
- Protocol and transport independence

Layered Programming Model

Layered Programming Model



What Does It Mean?

- Upper layer uses annotations extensively
  - Easy to use
  - Great toolability
  - Fewer generated classes
- Lower layer is more traditional
  - API-based
  - For advanced scenarios
- Most application will use the upper layer only
- Either way, portability is guaranteed

Server Side Programming: from WSDL & from POJO

Two ways to create a Web Service

- Starting from a WSDL file (top-down approach)
  - Generate classes using ***wsimport***
    - WS interface
    - WS implementation skeleton class
  - You add business logic to the WS implementation skeleton class
  - Build, deploy, and test
- Starting from a POJO (bottom-up approach)
  - Annotate POJO
  - Build and deploy
    - WSDL file generated automatically

Server-Side Programming Model: (Starting from POJO)

1. Write a POJO implementing the service
2. Add *@WebService* annotation to it
3. Optionally, inject a *WebServiceContext*
4. Deploy the application
5. Point your clients at the WSDL
   - e.g. *http://myserver/myapp/MyService?WSDL*

Example 1: Servlet-Based Endpoint

```
@WebService
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}
```

- @WebService annotation
  - All public methods become web service operations
- WSDL/Schema generated automatically
  - Default values are used

Example 2: EJB-Based Endpoint

```
@WebService
@Stateless
public class Calculator {
 @Resource
 WebServiceContext context;

    public int add(int a, int b) {
        return a+b;
    }
}
```

- It's a regular EJB 3.0 component, so it can use any EJB features
  - Transactions, security, interceptors...

Customizing through Annotations

```
@WebService(name="CreditRatingService",
            targetNamespace="http://example.org")
public class CreditRating {

    @WebMethod(operationName="getCreditScore")
    public Score getCredit(
        @WebParam(name="customer")
        Customer c) {
        // ... implementation code ...
    }
}
```

Demo

- Build a “Hello World” Web service using @WebService annotation
- Test the Web service
- Display the generated WSDL document

Client Side Programming: Java SE & Java EE

Java SE Client-Side Programming

1. Point a tool (NetBeans or wsimport) at the WSDL for the service

   ```
   wsimport http://example.org/calculator.wsdl
   ```

2. Generate annotated classes and interfaces
3. Call new on the service class
4. Get a proxy using a get<ServiceName>Port method of the service object
5. Invoke any remote operations

Example: Java SE-Based Client

```
CalculatorService svc =
                new CalculatorService();
Calculator proxy = svc.getCalculatorPort();
int answer = proxy.add(35, 7);
```

- No need to use factories
- The code is fully portable
- XML is completely hidden from programmer

Demo

- Build and run a Web service client of “Hello World” Web service using the WSDL document

Java EE Client-Side Programming

1. Point a tool (NetBeans or wsimport) at the WSDL for the service

```
wsimport http://example.org/calculator.wsdl
```

2. Generate annotated classes and interfaces
3. Inject a @WebServiceReference of the appropriate type
4. Invoke any remote operations

Example: Java EE-Based Client

```
@Stateless
public class MyBean {

    // Resource injection
    @WebServiceRef(CalculatorService.class)
    Calculator proxy;

    public int mymethod() {
        return proxy.add(35, 7);
    }
```

Annotations

Annotations Used in JAX-WS

- JSR 181: Web Services Metadata for the Java Platform
- JSR 222: Java Architecture for XML Binding (JAXB)
- JSR 224: Java API for XML Web Services (JAX-WS)
- JSR 250: Common Annotations for the Java Platform

@WebService

- Marks a Java class as implementing a Web Service, or a Java interface as defining a Web Service interface.
- Attributes
  - endpointInterface
  - name
  - portName
  - serviceName
  - targetNamespace
  - wsdlLocation

@WebMethod

- Customizes a method that is exposed as a Web Service operation
- The method is not required to throw java.rmi.RemoteException.
- Attributes
  - action
  - exclude
  - operationName

@WebParam

- Customizes the mapping of an individual parameter to a Web Service message part and XML element.
- Attributes
  - header
  - mode
  - name
  - partName
  - targetNamespace

Example

```
@WebService(targetNamespace =
   "http://duke.example.org", name="AddNumbers")
@SOAPBinding(style=SOAPBinding.Style.RPC,
   use=SOAPBinding.Use.LITERAL)
public interface AddNumbersIF {

   @WebMethod(operationName="add",
     action="urn:addNumbers")
   @WebResult(name="return")
   public int addNumbers(
      @WebParam(name="num1")int number1,
      @WebParam(name="num2")int number2) throws
        AddNumbersException;

}
```

Protocol and Transport Independence

Protocol and Transport Independence

- Typical application code is protocol-agnostic
- Default binding in use is SOAP 1.1/HTTP
- Server can specify a different binding, e.g.
  `@BindingType(SOAPBinding.SOAP12HTTP_BINDING)`
- Client must use binding specified in WSDL
- Bindings are extensible, expect to see more of them
  - e.g. SOAP/Java Message Service(JMS) or XML/SMTP

Example

```
@WebService
@BindingType(value=SOAPBinding.SOAP12HTTP_BINDING)
public class AddNumbersImpl {

    // More code
```

Catalog file

Why jax-ws-catalog.xml?

- One of the problems with using Netbeans to generate web services is that when you want to create a new Web service client, you are asked to select the web service which is normally on your local machine.
- This means that the web service has an address like http://localhost:8080 and it then becomes a problem when you deploy to other environments.
- One way around this is to use a catalog to resolve the addresses at run time. The file is called jax-ws-catalog.xml and it's placed in the META-INF directory.

Example: jax-ws-catalog.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
prefer="system">
    <rewriteSystem
            systemIdStartString="http://localhost:8080/"
            rewritePrefix="http://some url:some port/"/>
    <rewriteURI
             uriStartString="http://localhost:8080/"
             rewritePrefix="http://some url:some port/" />
</catalog>
```

Document/Literal Wrapped Convention

“Wrapped” document/literal

- Two parameter styles for document/literal
  - Wrapped (Default)
  - Bare
- Wrapped" is a form of document/literal, therefore it must follow all the rules defined for document/literal

doc/literal Rules

- When defining a document/literal service, there can be at most one body part in your input message and at most one body part in your output message.
- You do *not* define each method parameter as a separate part in the message definition.
- Each part definition must reference an element (not a type) defined, imported, or included in the types section of the WSDL document.

document/literal - XML schema

```
<?xml version="1.0" encoding="UTF-8"?><!-- Published by JAX-WS RI at http://jax-
ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.3-hudson-757-SNAPSHOT. -->
<xs:schema xmlns:tns="http://mypackage/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://mypackage/">

    <xs:element name="sayHello" type="tns:sayHello"></xs:element>

    <xs:element name="sayHelloResponse"
type="tns:sayHelloResponse"></xs:element>

    <xs:complexType name="sayHello">
        <xs:sequence>
            <xs:element name="arg0" type="xs:string" minOccurs="0"></xs:element>
            <xs:element name="arg1" type="xs:int"></xs:element>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="sayHelloResponse">
        <xs:sequence>
            <xs:element name="return" type="xs:string" minOccurs="0"></xs:element>
        </xs:sequence>

    </xs:complexType>
</xs:schema>
```

document/literal - WSDL

```
<types>
  <xsd:schema>
    <xsd:import namespace="http://mypackage/"
schemaLocation="http://localhost:8080/HelloWebService/HelloService?
xsd=1"></xsd:import>
  </xsd:schema>
</types>
<message name="sayHello">
  <part name="parameters" element="tns:sayHello"></part>
</message>
<message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse"></part>
</message>
<portType name="Hello">
  <operation name="sayHello">
    <input message="tns:sayHello"></input>
    <output message="tns:sayHelloResponse"></output>
  </operation>

</portType>
```

“Wrapped” Style

- A wrapper element must be defined as a complex type that is a sequence of elements. Each child element in that sequence will be generated as a parameter in the service interface.
- The name of the input wrapper element must be the same as the operation name.
- The name of the output wrapper element should be (but doesn't have to be) the operation name appended with "Response"

Java Code for SOAP Binding

```
package mypackage;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.ParameterStyle;

@WebService
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,  // default
                    use=SOAPBinding.Use.LITERAL,          // default
                    parameterStyle = ParameterStyle.WRAPPED) // default
public class Hello {
    public String sayHello(String name, int age){
        return "Hello " + name + " my age " + age;
    }
}
```

Java Code for SOAP Binding

```
package mypackage;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.ParameterStyle;

@WebService
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,  // default
                    use=SOAPBinding.Use.LITERAL,          // default
                    parameterStyle = ParameterStyle.BARE)
public class Hello {
    public String sayHello(String name, int age){   // Exception will occur
        return "Hello " + name + " my age " + age;
    }
}
```

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"