Web Services in Java

The shortest path to exposing and consuming web services in Java

Motivation

- Get web services up and running:
  - As quickly as possible
  - With as little overhead as possible
- Consume web services:
  - Easy API for the client

What kind of Web Service?

- “REST” Service:
  - No SOAP – just standard
  - Generally dependent on HTTP for transport
- WSDL/SOAP based:
  - More like RPC than document exchange
  - Can still do “document” style web services
  - Based on w3c standards
  - Transport mechanism independent

Web Services: State of the Art

- Java Web Services Developer Pack (reference implementation)
- Apache Axis
- Application Servers – most have web services functionality:
  - WS-I required by the J2EE 1.4 spec
  - WS-I = Web Services Interoperability organization – more info at http://www.ws-i.org/
- Codehaus XFire – the new kid on the block

Intro to Apache Axis

- Brief History:
  - IBM SOAP4J circa 2000
  - Donated and renamed Apache SOAP
  - Rebuilt as Apache Axis in 2002
- Latest version: Axis 1.3 (Oct. 5, 2005)
- De facto standard in the Java world
- Servlet-based

Exposing Web Services with Apache Axis

- Set up basic Axis webapp
- Generate WSDL from service class
- Generate deployment descriptor (WSDD) from the WSDL
- Update WSDD to point to the correct service implementation
- Call the AdminService to register the service

Setting up the Axis Webapp

- What you need:
  - libs
  - properties files
  - jsp's (optional, but recommended)

```
<war destfile="axisSample.war" webxml="etc/WEB-INF/web.xml">
    <fileset dir="src/jsp"/>
    <classes dir="etc/properties"/>
    <lib dir="lib"/>
</war>
```

- Test using happyaxis.jsp, also Version WS

Generating WSDL in Axis

- Use the java2wsdl ANT task:

```
<taskdef resource="axis-tasks.properties"
    classpathref="axis.classpath"/>

<target name="java2wsdl">
<axis-java2wsdl
 classname="com.chariotsolutions.wsinjava.axis.SampleService"
 namespace="axis.wsinjava.chariotsolutions.com"
 location="http://localhost:8080/axisSample/services/SampleService"
 output="axisSample.wsdl">
  <classpath path="bin"/>
</axis-java2wsdl>
</target>
```

Generating the WSDD Deployment Descriptor

- Use the wsdl2java ANT task:

```
<axis-wsdl2java
    output="gen/java"
    verbose="true"
    url="gen/wsdl/axisSample.wsdl"
    serverside="true"/>
```

- This task also builds the client-side classes

Fixing the WSDD Deployment Descriptor

- The WSDD as generated will include a reference to the BindingStubImpl:

```
<service ...>
  <parameter name="className"
    value="com.chariotsolutions.wsinjava.axis.
    SampleServiceBindingStubImpl"/>
  </service>
```

- Fix this within ANT:

```
<replaceregexp
   file="gen/java/com/chariotsolutions/wsinjava/axis/deploy.wsdd"
   match="SoapBindingImpl"
   replace=""/>
```

Calling the Axis AdminService

- Use the axis-admin ANT task to call the Axis AdminService to register our new service:

```
<axis-admin
  failonerror="true"
  servletpath="/axisSample/services/AdminService"
  debug="true"
  xmlfile="gen/java/com/chariotsolutions/wsinjava/axis/deploy.wsdd"
/>
```

- On success, our service is configured with Axis and should be ready to use

Testing the service

- Axis will automatically regenerate the WSDL for the service by appending the “wsdl” parameter to the web request:

http://localhost:8080/axisSample/services/SampleService?wsdl

- You can test service methods by appending a “method” parameter to the web request:

http://localhost:8080/axisSample/services/SampleService?method=getTheMeaningOfLife

Testing the service: method response

- Here's the SOAP response:

```
<soapenv:Envelope>
<soapenv:Body>
  <getTheMeaningOfLifeResponse
      soapenv:encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/">
      <getTheMeaningOfLifeReturn href="#id0"/>
  </getTheMeaningOfLifeResponse>
  <multiRef id="id0" soapenc:root="0" soapenv:encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="xsd:int">42</multiRef>
</soapenv:Body>
</soapenv:Envelope>
```

Document Style Web Services in Axis

- RPC style services are the default in Axis
- Axis uses SOAP for **all** service types
- Other service types:
  - <u>Document</u> services do not use any encoding
  - <u>Wrapped</u> service "unwrap" the SOAP body into individual parameters.
  - <u>Message</u> services receive and return arbitrary XML in the SOAP Envelope without any type mapping / data binding.

Other Service Types

```
<soap:Envelope xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <soap:Body>
    <myNS:PurchaseOrder xmlns:myNS="http://commerce.com/PO">
      <item>SK001</item>
      <quantity>1</quantity>
      <description>Sushi Knife</description>
    </myNS:PurchaseOrder>
  </soap:Body>
</soap:Envelope>
```

- Document Style:
  `public void method(PurchaseOrder po);`

- Wrapped Style:
  `public void purchaseOrder(String item, int quantity, String desc)`

Axis Message Type

- DOM Elements or SOAPBodyElements:
  - public Element [] method(Element [] bodies);
  - public SOAPBodyElement [] method (SOAPBodyElement [] bodies);
- DOM Document representing the soap body:
  - public Document method(Document body);
- SOAPEnvelopeObject
  - public void method(SOAPEnvelope req, SOAPEnvelope resp);

Intro to JBossWS

- JBossWS is JBoss's implementation of web services
- Brief History:
  - JBoss.NET: earier implementation JBoss 3.x
  - Replaced by JBossWS in JBoss 4.x
- Currently based on a modified axis-1.1 under the hood
- Plans to replace with JBoss's own SOAP stack

More about JBossWS

- As with Axis, servlet-based
- Can expose POJOs and Stateless Session Beans
  (= “Java” endpoints or EJB endpoints)

Exposing Web Services with JBossWS

- Generate WSDL and jaxrpc-mapping.xml from service implementation
- Create web.xml by hand
- Create an additional deployment descriptor: webservices.xml

Generate WSDL and Mappings

- JBoss docs recommend wscompile ANT task from JWSDP:

```
<wscompile base="gen" f="rpcliteral" mapping="gen/etc/jaxrpc
      mapping.xml" server="true"
      config="etc/wscompile/config.xml" fork="true" debug="true">
      <classpath>
            <path refid="jwsdp.classpath"/>
            <path location="bin"/>
      </classpath>
</wscompile>
```

Create web.xml by hand

```
<web-app>

<servlet>
   <servlet-name>JBossSample</servlet-name>
   <servlet-class>
com.chariotsolutions...JBossWSSampleJSEEndpoint
   </servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>JBossSample</servlet-name>
   <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

Create webservices.xml

```
 <webservice-description>
    <webservice-description-name>
JbossSampleService
     </webservice-description-name>
    <wsdl-file>
WEB-INF/wsdl/JBossWSSampleService.wsdl
     </wsdl-file>
    <jaxrpc-mapping-file>
WEB-INF/jaxrpc-mapping.xml
</jaxrpc-mapping-file>
    <port-component>
      <wsdl-port>JBossWSSamplePort</wsdl-port>
      <service-endpoint-interface>
com.chariotsolutions.wsinjava.jbossws.JBossWSSample
        </service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>JBossSample</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

Intro to Codehaus XFire

- Relative newcomer, announced Aug 2004
- Like Axis, also Servlet-based
- Choice of XML Binding:
  - XMLBeans: see xmlbean.apache.org
  - Aegis: simple XML binding
- Choice of transport protocol: HTTP or XMPP (Jabber protocol)

Exposing Web Services with XFire

- Set up basic XFire webapp
- Configure your services in services.xml

Set up basic XFire Webapp

- All you need:
  - libs
  - web.xml
  - services.xml

```
<war destfile="xFireSample.war" webxml="etc/WEB-INF/web.xml">
    <fileset dir="etc" excludes="WEB-INF/**"/>
    <lib dir="lib"/>
</war>
```

Configure your services in services.xml

- Couldn't be simpler:

```
<xfire>
  <services>
    <service>
      <serviceClass>
            com.chariotsolutions.wsinjava.xFire.SampleService
      </serviceClass>
    </service>
  </services>
</xfire>
```

Consuming Web Services

- Can create client-side versions of our service object using:
  - Axis: the *wsdl2java* ANT task
  - JWSDP: the *wscompile* ANT task
- XFire: client generation experimental right now: “This will probably not work for you.”

Wsdl2java Task

```
<axis-wsdl2java
    output="gen/java"
    verbose="true"
    url="http://localhost:8080/axisSample/services/AxisSample?wsdl"
    serverside="false"/>

<axis-wsdl2java
 output="gen/java"
 verbose="true"
 url="http://localhost:8080/xFireSample/services/XFireSample?wsdl"
 serverside="false"/>
```

Axis Client for Axis Server

- Just works:
  - Complex types, arrays
  - Date becomes java.util.GregorianCalendar
  - Application exceptions are passed through

Axis Client for XFire Server

- Works: complex types, arrays
- XFire WSDL does not create new types for user exceptions, but the message comes through:

```
try {
	service.getException();
}
catch(RemoteException ex) {
	System.out.println("got RemoteException: " + ex.getMessage());
}
```

References

- Axis:
  - http://ws.apache.org/axis/java/user-guide.html
  - http://www.lucianofiandesio.com/javatales/axant.html
- XFire:
  - http://xfire.codehaus.org/User's+Guide
- JBoss
  - http://www.jboss.org/wiki/Wiki.jsp?page=JBossWS

Download Slides and Code

- Slides and Code can be downloaded from:

  www.chariotsolutions.com

- Further questions:

  jhammen@gmail.com