A Tutorial for New Axis2 Users

Dong Liu

edongliu@gmail.com

MADMUC, Univ. of Saskatchewan

Outline

- Suggested prerequisite reading
- Install Axis2
- Understand Axis2 architecture
- Hello, Axis2
- Addition service
- Want to know More?

Suggested prerequisite reading

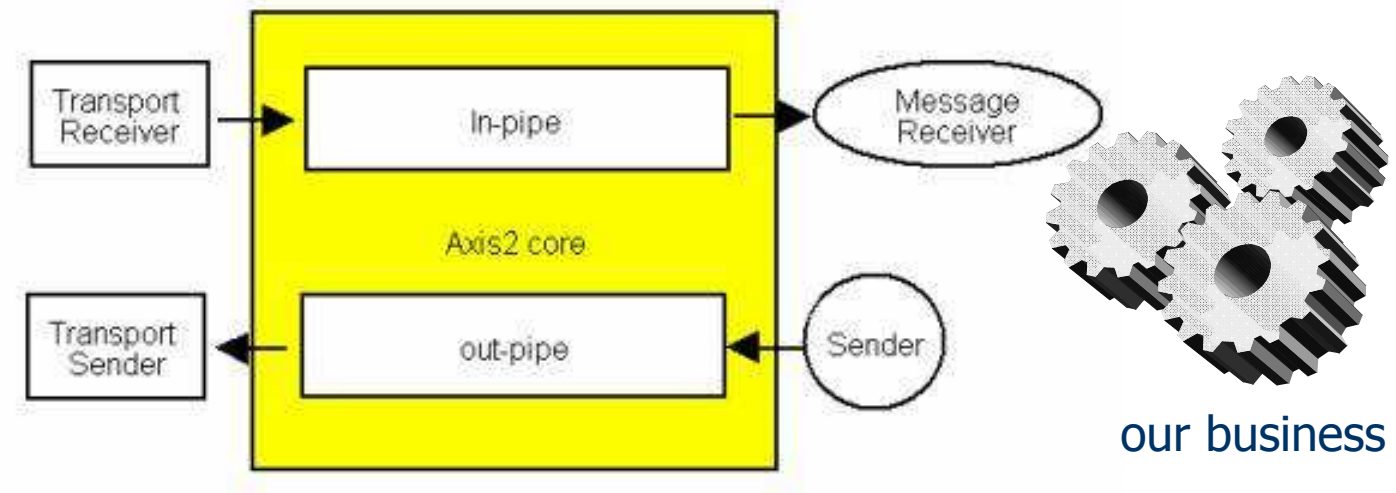
- If you are not familiar with XML and XML schema, please go through the following courses
  - XML Tutorial http://www.w3schools.com/xml/default.asp or http://www-128.ibm.com/developerworks/xml/newto/index.html
  - XML Schema Tutorial http://www.w3schools.com/schema/default.asp
  - SOAP Tutorial http://www.w3schools.com/soap/default.asp
  - WSDL Tutorial http://www.w3schools.com/wsdl/default.asp
- If you use command line and are not familiar with the Java tools like javac and jar, please read this tutorial http://www.herongyang.com/jtool/
- If you want to know more about WSDL and SOAP, you can refer to
  - Web Services Description Language (WSDL) Version 2.0 Primer http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/
  - W3C SOAP Version 1.2 Primer http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

Install Axis2

- Assume that you have Tomcat installed in your system (call the place catalina_home), and you can access its service at http://localhost:8080
  - Hint: check if you have the native library installed (on Windows, it is the tcnative*.dll file in catalina_home\bin). If not, you can download it from http://tomcat.heanet.ie/native/ It will increase your tomcat performance.
- Download the newest version of Axis2
  - Get the War distribution for deployment on Tomcat
  - Get the Standard distribution for service and client programming
  - Get the Documents distribution for Java API descriptions
- Installation
  - Installation of Axis2 on Tomcat means you drop the .war release at catalina_home\webapps , and restart Tomcat. Make sure you see the welcome page at http://localhost:8080/axis2/ , and pass the validation. Note: even if you get success when validating the installation, you may still miss some java packages.
  - Installation of Axis2 on your programming environment means you unzip the standard distribution to somewhere in your system, say axis2_home, and add all .jar files in axis2_home\lib into your java class path. If you use a java IDE, you may look at the java build path for the project. If you work in command line, you may define an environment variable named, say, axis2_cp.

Understand Axis2 architecture

- Tomcat works as a servlet container, and the deployed Axis2 War is a servlet
- Axis2 is a soap engine: It processes the soap messages in and out http://localhost:8080/axis2/services/*
- Axis2 will take care of addressing, security, reliable messaging, and others; the service developer needs to take care of the business behind the Message Receiver and before the Sender



http://www.developer.com/java/ent/article.php/10933_3606466_1

Hello, Axis2 – the problem

- Develop a web service that replies 'hello, the_name' when we send to it a message with 'the_name'
- Generally, there are two ways to develop services, top-down and bottom-up, or schema/wsdl-first and code-first. Personally, I prefer developing the documents first
- The first step is to write the wsdl file for this service
  - wsdl file is the contract between the service provider and service requestors

Hello, Axis2 – ..\helloAxis2\helloAxis2.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.example.org/helloAxis2/"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            name="helloAxis2" targetNamespace="http://www.example.org/helloAxis2/">
     <wsdl:types>
          <xsd:schema targetNamespace="http://www.example.org/helloAxis2/">
               <xsd:element name="Response" type="xsd:string" />
               <xsd:element name="Request" type="xsd:string" />
          </xsd:schema>
     </wsdl:types>
     <wsdl:message name="Response">
          <wsdl:part element="tns:Response" name="Response" />
     </wsdl:message>
     <wsdl:message name="Request">
          <wsdl:part element="tns:Request" name="Request" />
     </wsdl:message>
     <wsdl:portType name="helloAxis2">
          <wsdl:operation name="sayHello">
               <wsdl:input message="tns:Request" />
               <wsdl:output message="tns:Response" />
          </wsdl:operation>
     </wsdl:portType>
     <wsdl:binding name="helloAxis2SOAP" type="tns:helloAxis2">
          <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
          <wsdl:operation name="sayHello">
               <soap:operation soapAction="http://localhost:8080/axis2/services/helloAxis2" />
               <wsdl:input>
                    <soap:body use="literal" />
               </wsdl:input>
               <wsdl:output>
                    <soap:body use="literal" />
               </wsdl:output>
          </wsdl:operation>
     </wsdl:binding>
     <wsdl:service name="helloAxis2">
          <wsdl:port binding="tns:helloAxis2SOAP" name="helloAxis2SOAP">
               <soap:address location="http://localhost:8080/axis2/services/helloAxis2" />
          </wsdl:port>
     </wsdl:service>
</wsdl:definitions>
```

Hello, Axis2 – code generation

- Axis2 can generate the service skeletons (server side code), the service stubs (client side code), the service description (services.xml), and others
- Two ways to generate code
  - Use command line or Ant task
  - Use the plug-in for your IDE (Eclipse or intelliJ IDEA)
- Run the following command line to generate the skeletons, and services.xml
- We have 3 files now: HelloAxis2MessageReceiverInOut.java is the message receiver, HelloAxis2Skeleton.java is the skeleton, and services.xml is the service description

```
..\helloAxis2>java -cp %axis2_cp% org.apache.axis2.wsdl.WSDL2Java
-uri helloAxis2.wsdl -ss -sd -s -d none
```

`..\helloAxis2` : where you save your wsdl file
`axis2_cp` : the environment variable we have discussed
`-s` : generate code only for sync style
`-d none` : I don't use any data binding framework
The details of code generation tool options is available at
http://ws.apache.org/axis2/tools/1_0/CodegenToolReference.html

Hello, Axis2 – coding 1

- Nothing needs to be changed in HelloAxis2MessageReceiverInOut.java
- Add the logic in HelloAxis2Skeleton.java
  - Axiom is the object model for xml processing in Axis2, http://ws.apache.org/commons/axiom/OMTutorial.html
  - Get the information you need from the OMElement that the Message Reciever passes to the skeleton, in this case, "the_name"
  - Generate the response OMElement with "hello, the_name", and return it
- Parse and construct xml object according to the request and response schema in slide 7

Hello, Axis2 – coding 2

List of HelloAxis2Skeleton.java

```
package helloAxis2;

import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;

public class HelloAxis2Skeleton {
      public org.apache.axiom.om.OMElement sayHello(
             org.apache.axiom.om.OMElement param0)
      {
            OMFactory fac = OMAbstractFactory.getOMFactory();
            //create a namespace
            OMNamespace omNs = fac.createOMNamespace(
                              "http://www.example.org/helloAxis2/", "helloAxis2");
            //create an element using the namespace
            OMElement response = fac.createOMElement("Reponse", omNs);
            //get the text content of request message, compose the content of
            //response message, and add it to the element
            response.addChild(fac.createOMText("Hello, " + param0.getText()));
            return response;
      }
}
```

Hello, Axis2 – deployment

- Archive the files into helloAxis2.aar
  - .aar is the same as .zip or .jar, so you can use jar, or zip to create the archive, and change the file extension to aar
  - The archive should include all the server side class files and a folder named META-INF with the wsdl and services.xml files inside
  - In this case, the structure is

```
helloAxis2.aar
|
+---helloAxis2
|       HelloAxis2MessageReceiverInOut.class
|       HelloAxis2Skeleton.class
|
\---META-INF
        helloAxis2.wsdl
        services.xml
```

- Drop the .aar file to catalina_home\webapps\axis2\WEB-INF\services
- Restart Tomcat, and check the new service added

Hello, Axis2 – testing 1

- Two ways to test the service
  - Develop a specific web service client
  - Use a general HTTP client and specify the http message
- Axis2 has a client package

List of helloClient.java

```
package helloAxis2;

import java.io.StringWriter;
import javax.xml.stream.*;
import org.apache.axiom.om.*;
import org.apache.axis2.*;

public class helloClient {
      //specify the endpoint reference
      private static EndpointReference targetEPR = new EndpointReference(
                  "http://localhost:8080/axis2/services/helloAxis2");
      public static void main(String[] args) {
            try {
                  OMFactory fac = OMAbstractFactory.getOMFactory();
                  OMNamespace omNs = fac.createOMNamespace(
                        "http://www.example.org/helloAxis2/", "helloAxis2");
                  //compose the request element
                  OMElement request = fac.createOMElement("Request", omNs);
                  request.addChild(fac.createOMText("MADMUC"));
                  …
```

Hello, Axis2 – testing 2

List of helloClient.java, continued

```
            Options options = new Options();
            //specify sender options: target, transport protocol, and soap action
            options.setTo(targetEPR);
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);
            options.setAction("http://localhost:8080/axis2/services/helloAxis2");

            ServiceClient sender = new ServiceClient();
            sender.setOptions(options);
            OMElement result = sender.sendReceive(request);

            //print out the response
            StringWriter writer = new StringWriter();
            result.serialize(XMLOutputFactory.newInstance()
                                .createXMLStreamWriter(writer));
            writer.flush();
            System.out.println(writer.toString());
        } catch (AxisFault axisFault) {
            axisFault.printStackTrace();
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
    }
}
```

Run the client, and you will see the response

```
<helloAxis2:Reponse xmlns:helloAxis2="http://www.example.org/helloAxis2/">
Hello, MADMUC</helloAxis2:Reponse>
```

Hello, Axis2 – code revisit

Check services.xml to understand what happened

```
<serviceGroup>
     <service name="helloAxis2">
          <messageReceivers>
               <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
                    class="helloAxis2.HelloAxis2MessageReceiverInOut" />
          </messageReceivers>
          <parameter name="ServiceClass" locked="false">
               helloAxis2.HelloAxis2Skeleton
          </parameter>
          <operation name="sayHello"
               mep="http://www.w3.org/2004/08/wsdl/in-out">
               <actionMapping>
                    http://localhost:8080/axis2/services/helloAxis2
               </actionMapping>
          </operation>
     </service>
</serviceGroup>
```

Axis2 calls this message receiver for the requests

Then the receiver will call this class

Axis2 interprets the SOAPAction by this

Hello, Axis2 – http flows 1

It is better to check the http flows when testing and debugging services and clients. You can use the tcpmon tool from Axis (not Axis2) to do this. Read http://ws.apache.org/axis/java/user-guide.html#AppendixUsingTheAxisTCPMonitorTcpmon for more information

Request

```
POST /axis2/services/helloAxis2 HTTP/1.1
User-Agent: Axis/2.0
SOAPAction: http://localhost:8080/axis2/services/helloAxis2
Host: 127.0.0.1:8081
Transfer-Encoding: chunked
Content-Type: text/xml; charset=UTF-8

119
<?xml version='1.0' encoding='UTF-8'?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header />
      <soapenv:Body>
         <helloAxis2:Request xmlns:helloAxis2="http://www.example.org/helloAxis2/">guess</helloAxis2:Request>
      </soapenv:Body>
   </soapenv:Envelope>0
```

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=D4C7801FBEAA460B4CEF08B9BE430D13; Path=/axis2
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 27 Sep 2006 04:05:12 GMT

120
<?xml version='1.0' encoding='UTF-8'?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header />
      <soapenv:Body>
         <helloAxis2:Reponse xmlns:helloAxis2="http://www.example.org/helloAxis2/">Hello, MADMUC</helloAxis2:Reponse>
      </soapenv:Body>
   </soapenv:Envelope>
0
```

Addition service – the problem

- The service get a message containing two integers and give the sum in response
- Suppose the request and response messages are like the following

```
<add>
   <number>1</number>
   <number>2</number>
</add>
```

Request

```
<sum>3</sum>
```

Response

- This time, we start directly from coding in case that you do not like the long wsdl file

Addition service – coding 1

- Axis2 supplies a basic message receiver, org.apache.axis2.receivers.RawXMLINOutMessageReceiver, that parses the first XML element in SOAP Envelope/Body, and call the operation corresponding to the SOAPAction by passing the XML element to it
- What we need to do is write the service class, and configure the service in services.xml

List of addition.java

```
package additionService;

import java.util.Iterator;
import org.apache.axiom.om.*;
public class addition {
     public OMElement add(OMElement param0)
     {
        OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs = fac.createOMNamespace(
                "http://www.example.org/additionService/", "additionService");
        OMElement sum = fac.createOMElement("sum", omNs);
        Iterator num = param0.getChildren();
        int result = Integer.parseInt(((OMElement)num.next()).getText())
                        +  Integer.parseInt(((OMElement)num.next()).getText());
        sum.addChild(fac.createOMText("" + result));
        return sum;
     }
}
```

Addition service – coding 2

List of services.xml

```
<serviceGroup>
     <service name="additionService">
          <messageReceivers>
               <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
               class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver" />
          </messageReceivers>
          <parameter name="ServiceClass" locked="false">
               additionService.addition
          </parameter>
          <operation name="add"
               mep="http://www.w3.org/2004/08/wsdl/in-out">
               <actionMapping>
                    http://localhost:8080/axis2/services/additionService
               </actionMapping>
          </operation>
     </service>
</serviceGroup>
```

Use the message receiver provided in Axis2

Set the service class

Map SOAPAction to the operation

Addition service – Deployment and testing 1

- Archive the files into additionService.aar like we did for helloAxis2
- Drop the aar to the same place, restart Tomcat, and the service should run
- Testing the service

List of addClient.java

```
package additionService;

import java.io.StringWriter;
import javax.xml.stream.*;
import org.apache.axiom.om.*;
import org.apache.axis2.*;
public class addClient {
     private static EndpointReference targetEPR = new EndpointReference(
               "http://localhost:8080/axis2/services/additionService");

     public static void main(String[] args) {
          try {
               OMFactory fac = OMAbstractFactory.getOMFactory();
               OMNamespace omNs = fac.createOMNamespace(
                         "http://www.example.org/additionService/", "additionService");
               OMElement add = fac.createOMElement("add", omNs);
               OMElement number1 = fac.createOMElement("number", omNs);
               number1.addChild(fac.createOMText("1"));
               add.addChild(number1);
               OMElement number2 = fac.createOMElement("number", omNs);
               number2.addChild(fac.createOMText("2"));
               add.addChild(number2);
               …
```

Addition service – Deployment and testing 2

List of addClient.java, continued

```
        Options options = new Options();
        options.setTo(targetEPR);
        options.setTransportInProtocol(Constants.TRANSPORT_HTTP);
        options.setAction("http://localhost:8080/axis2/services/additionService");

        ServiceClient sender = new ServiceClient();
        sender.setOptions(options);
        OMElement result = sender.sendReceive(add);

        StringWriter writer = new StringWriter();
        result.serialize(XMLOutputFactory.newInstance()
                  .createXMLStreamWriter(writer));
        writer.flush();
        System.out.println(writer.toString());

    } catch (AxisFault axisFault) {
        axisFault.printStackTrace();
    } catch (XMLStreamException e) {
        e.printStackTrace();
    }
  }
}
```

Run the client, and you will see the response

```
<additionService:sum xmlns:additionService="http://www.example.org/additionService/"
xmlns:tns="http://ws.apache.org/axis2">3</additionService:sum>
```

Want to know more?

- Read Axis2 user guide http://ws.apache.org/axis2/1_0/userguide.html
- Ask at Axis mailing lists http://ws.apache.org/axis2/mail-lists.html
- Check the examples at axis2_home\samples
- Other web services framework
  - XFire http://xfire.codehaus.org/
  - Sun Java web services http://java.sun.com/webservices/index.jsp