Real World Axis2/Java: Highlighting Performance and Scalability

Deepal Jayasinghe
WSO2 Inc. & Apache Software Foundation
deepal@wso2.com
About the Presenter

• Technical Lead WSO2 Inc.
  – www.wso2.com
  – A start-up company aiming to develop and support leading edge Open Source Web service software
    • Provides support, training and consultancy for Web services
• Apache Software Foundation
  – Apache member
  – WS-PMC Member
  – Contributes to a number of WS projects
• Frequent speaker at ApacheCon and SOA related conferences
• Axis2 developer since day one.
What is Axis2?

- An Apache project
  - Freely available under the Apache License
- A Web Service toolkit
  - Handles SOAP and XML/HTTP
  - Maps XML to Java
    - Either built-in or using a binding framework
  - Supports different exchange patterns
- Extensible
  - Plug-in projects support Security (Rampart) and Reliability (Sandesha)
Agenda

- Axis2 client API
- Web service
- Service extension – Module
- Pluggable components
- Tools
- MTOM
- REST
Axis2 Client API

- There are two main APIs
  - `ServiceClient`
  - `OperationClient`

- In addition, there are two more convenient APIs as well
  - `Stub`
  - `RPCServiceClient`
ServiceClient

- Designed for average users
- Supports XML in XML
  - Takes input as AXIOM and returns AXIOM
- Inbuilt support for invoking all eight MEPs defined in WSDL 2.0
- Support for both Synchronous and Asynchronous invocation
- Many options to create a client
ServiceClient APIs

- ServiceClient has the following set of APIs
  - SendRobust
  - FireAndForget
  - SendReceive
  - SendReceiveNonBlocking
Adding Custom SOAP Headers

- There are three main ways of adding a header
  - Adding as a string header
    - addStringHeader(QName headerName, String headerText)
  - Adding as a SOAP header block
    - addHeader(SOAPHeaderBlock header)
  - Adding as an AXIOM element
    - addHeader(OMElement header)
What is Option object?

- Option object is used to configure the client
- HTTP properties
  - Chunking
  - Proxies
  - Authentication (NTLM, Basic, Digest)
  - Timeouts (Socket, Connection)
  - Support GZIP
  - HTTP Headers, etc.
- WS-Addressing properties (wsa:to, was:replyTo)
- Generic properties (name, value pairs)
Asynchronous Invocation

- There are two types
  - Application level
  - Transport level
- Uses the callback mechanism
- Provides the implementation object of AxisCallback, which has the following methods
  - OnMessage
  - OnFault
  - OnError
  - onComplete
Asynchronous

- WS-Addressing implies an asynchronous model
  - Even over a 'synchronous' transport like HTTP
**ServiceClient and Proxy**

- When we want to use a Proxy, then we have to set the Proxy object in options

```java
Options option = new Options();


proxy.setProxyName("localhost");

proxy.setProxyPort(2222);

option.setProperty(HTTPConstants.PROXY, proxy);
```
Advanced APIs

- Engaging a module
  - `sc.engageModule(moduleName);`
- Accessing the last OperationContext
  - `sc.getLastOperationContext();`
- Creating an OperationClient
  - `sc.createClient(QName operationQName)`
- Cleaning transport senders
  - `sc.cleanupTransport();`
OperationClient

- Designed for advanced users
- Has more control over incoming and outgoing messages
- Need to perform additional steps
  - Create SOAPEnvelop
  - Create a MessageContext
RPCServiceClient

- Convenient API for invoking a service
- An understanding of AXIOM is not required
- Extends from ServiceClient and has the following additional methods
  - OMElement invokeBlocking(QName opName, Object [] args)
  - Object[] invokeBlocking(QName opName, Object [] args, Class [] returnTypes)
  - invokeNonBlocking(QName opName, Object [] args, AxisCallback callback)
  - invokeRobust(QName opName, Object [] args)
RPCServiceClient client = new RPCServiceClient();
Options options = new Options();
options.setTo(new EndpointReference("http://localhost:8080/axis2/service/EchoService");
options.setAction("urn:echoString");
client.invokeBlocking(new QName("echoString"), new Object[]{"Hello"});

Object array may contains any kind of Java bean
Creating a client on the fly

No need to provide an option object

Can use either ServiceClient or RPCServiceClient

ServiceClient client = new ServiceClient(null,new URL("http://localhost:8080/axis2/services/EchoService?wsdl"),null,null);

OMElement response = client.sendReceive(new QName("http://ws.apache.org/axis2","sayHello"),null);
Quick Recap- WSDL

- WSDL is the Description Language for Web Services
  - kind of like IDL
- Three aspects
  - Interface
    - XML Schema for input message, output message
  - Binding
    - How this message fits into SOAP – e.g. SOAPAction Headers
  - Address
    - The URL of the endpoint this service is available on
Create a Client Using a WSDL

- First ensure you have
  - `SET AXIS2_HOME=c:\axis2-1.3`
  - or similar (EXPORT, SETENV, etc.)
- Command-line tool:
  - `WSDL2Java -uw -s -uri http://localhost:6060/axis2/services/EchoService?`
- Generates
  - Ant Build script – `build.xml`
  - `src\EchoServiceStub.java`
Stub

- Generate a client stub for a given WSDL
- Then use the generated stub to invoke the service

```java
EchoServiceStub stub = new EchoServiceStub();
EchoServiceStub.SayHelloResponse response = stub.sayHello();
System.out.println(response.get_return());
```

Output: Hello World!!!

- How to re-target a stub to a new EPR
  
  ```java
  stub = new EchoServiceStub(“http://mysere.com:8080/axis2/services/Echo”);
  ```
Client-side Session Handling

- ServiceClient supports invoking a service in a session aware manner
  - In the case of a SOAP session, it will copy the required reference parameter
  - In the case of a transport session, it will copy the HTTP cookies
- Axis2, by default, copies the JSESSION cookie ID
- If you want to change a custom cookie, then we need to specify the cookie name

```java
options.setProperty("customCookieID","SESSIONID");
```
Axis2 Web Service

- There are many methods of deploying a service
  - Using a service archive file
  - As a POJO
    - JSR 181 Annotated class or not
  - Programmatically
Let’s start with a POJO

```java
public class WeekdayFromDate {
    public String getWeekday(int year, int month, int day) {
        ...
    }
}
```
import java.io.IOException;
import org.apache.axis2.engine.AxisServer;

public class Main {
    public static void main(String[] args) throws IOException {
        AxisServer axisServer = new AxisServer();
        axisServer.deployService(pojo.WeekDayFromDate.class.getName());
        System.out.print("Hit Enter to stop");
        System.in.read();
        axisServer.stop();
    }
}
Demonstration

- http://localhost:6060

**Deployed services**

**WeekdayFromDate**

Available operations

- getWeekday
What happened?

- Axis2 has a simple built-in web server
  - SimpleHttpServer
- Started when deployService is called
- Default port is 6060
- Translated the GET parameters into the call against the POJO
- Translated the response value into an XML body
POJO Deployment

- Write a Java class
- Compile it
- Deploy into repository/pojos directory

- Has limited control
- A few issues when there are multiple classes
The Usual Model

- Normally, services are deployed in a **Axis2 ARchive (AAR)**
- Conceptually the same as a **WAR file**
- Deployment descriptor is called **META-INF/services.xml**
- Place the AAR file into the **Axis2 repository/services directory**
- **Axis2 hosted in Tomcat, etc. or standalone**
Service Archive

- Self contained package
  - Configuration files
  - Class files
  - Third party resources
  - WSDL and Schema files
  - Web resources

- Many configuration options

- Service isolation via one class loader per service
Services.xml

- One or more services in a single file
- Service level parameters
- Service class
- Service scope
- Configuration options for WSDL generation
  - Targetnamespace
  - Schema targetNamespace
- Expose and Exclude operations
- Expose transports
- WS-Policy
<service name="HelloService">
  <description>
    POJO: Hello Service
  </description>
  <messageReceivers>
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
      class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
      class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>
  <parameter name="ServiceClass">Hello</parameter>
</service>
Associate WSDL with a Service

- We can have one or more WSDL files inside META-INF
- To associate a WSDL with a service
  - The WSDL service name should be equal to the service name in the services.xml
- When we want to display our own WSDL

<parameter name="useOriginalwsdl">true</parameter>
Session Aware Service

- There are four types
  - Application
  - Transport
  - SOAP Session
  - Request

- Specifying a service scope

  `<service scope="application">  </service>`

- Accessing contexts
AxisService (Continued)

- ServiceLifeCycle interface
  - Managing the life cycle of a service
- LifeCycle interface
  - Managing a session life cycle
- Creating a new class inside a service
  - Need to know about Axis2 class loader concept
Adding a Web Resource

- Services can be deployed alone with a web resource (jsp, js, html, etc.)
- When we have a web resource, we have to put them under the directory called 'WWW'
  
  META-INF
  
  services.xml

  WWW
  
  index.html

- Upon deployment, you can access the resource using (if the service archive name is foo.aar)

  http://localhost:8080/axis2/foo
Axis2 Module

- Modules are how Axis2 extends or provides quality of services
- Implementation of WS-Specification
- Can add support for a new specification in a simple and clean manner
- Can deploy as a self contained package
- Has own class loader module, so the module is also isolated
AxisModule (Continued)

- Location of handlers can be configured using phase rules
- Available phases are specified in axis2.xml
- Phase orders are not likely to be changed
- There are four types of execution chains

InFlow, OutFlow, InFaultFlow and OutFaultFlow
module.xml

- module.xml contains
  - Parameters
  - Handlers + Phase rules
  - WS-Policy
  - Module implementation class
  - End points
Axis2 modules

- Addressing – WS-Addressing
- Rampart – WS-Security and WS-SecureConversation
- Rahas – WS-Trust
- Sandesha – WS-ReliableMessaging
- Logging – logs messages
- Mex – supports WS-MetadataExchange
- SOAPMonitor – allows you to trace SOAP messages on the fly
- Ping – allows you to test if a service is alive without calling any business methods
AxisModule (Continued)

- Module can be deployed as
  - an archive file
  - a directory

- Modules are not hot deployable as services

- Module has two states
  - Available: available for engaging
    - No handlers are in the chains
  - Engaged: engaged to Axis2
    - Handlers are in the chains
Module, Phase and Handlers

Module

InFlow

H1

H2

Outlow

H3

H1

H2

H3
Axis2 Code Generation

- There are many options for data binding frameworks
  - ADB
  - XMLbeans
  - JaxBri
  - JibX
- We can easily add a new data binding framework
Code Generation (Continued)

- Support for client side and server side code generation
- Support for both WSDL 1.1 and 2.0
  - Support for REST client generation
- Client side stub can be configured using WS-Policy
- Client stub can be generated for sync or async
In the server side, it generates

- Data bound classes
- Custom message receivers
- Service skeleton
- services.xml
- Copy of original WSDL
- Ant build file to generate service archive file
Code Generation (Continued)

- Tools available for code generation
  - IDE plugins (Both Eclipse and IntelliJ idea)
  - Command line tools
- WSO2 WSAS has a convenient way of creating either client or server side code
Code Generation Options

• Usage WSDL2Code

- uri <Location of WSDL> : WSDL file location
- o <output Location> : output file location
- a : Generate async style code only. Default is off
- s : Generate sync style code only. Default is off. takes precedence over -a
- p <package name> : set custom package name
- l <language> : valid languages are java and csharp. Default is java
- t : Generate Test Case to test the generated code
- ss : Generate server side code (i.e. skeletons). Default is off
- sd : Generate service descriptor (i.e. axis2.xml). Default is off. Valid with –ss

-d: choose databinding model – adb, jaxb, xmlbeans, none

There are many more data binding options
Java2WSDL

- Generating a WSDL file from a Java class
- We can generate
  - WSDL 2.0
  - WSDL 1.1
    - doclit/wrapped
    - doclit/bare
- Can be used as a starting point of a complex system
- Available tools
  - IDE plugins
  - Command line tools
Axis2 Pluggable Components

- Extending Axis2 deployment
  - Custom deployers
- Axis2 observers
  - WSO2 WSAS atom and RSS generation
- Custom message receivers
  - Talking to non-java class
- Adding a new transport
Custom Deployers

- A convenient way to make any system, including legacy systems, a Web service
  - Exposing a database as a Web service
  - Java Script as a Web service
  - Deploying an Axis1 service in Axis2
- Easily pluggable via axis2.xml
- Have to implement org.apache.axis2.deployment.Deployer
Data Service

- Unlock legacy relational data into your SOA or Web 2.0 architecture
- Zero-code programming model allowing rapid development
- Ability to use full power of the WS-* security stack to ensure data security
Axis1 Deployer

- Ability to deploy any Axis1 service in Axis2
- Uses Axis2 features as it is
  - Hot deployment, hot update
- Zero additional code required
Writing Our Own Deployer

- There are a few things when developing a new deployer
  - How to make something into an Axis2 component
    - Service, Module or anything else
  - How to remove that
  - How to populate an object correctly

- As an exercise, write a deployer to make a pure Java class into a service
Axis2 Observers

- Observing what happened inside AxisConfiguration
  - Adding service
  - Changing service
  - Adding module
  - Changing module
- Observer will notify when the above events happen
- Then the observer can act upon that
  - WSO2 WASA atom and RSS feed generation
Custom Message Receivers

- A message receiver is MEP dependent
- A message receiver can be considered as the last handler of the in chain
- You can write a Message receiver
  - Invoke non Java class
    - Java Script, Groovy, etc.
  - As a service
    - You can deploy a service only with a MR
Message Receivers

- `<S:Body>`
  - RPC MessageReceiver
  - Generic ADB XML->Java
  - POJO

- `<S:Body>`
  - Custom MessageReceiver
  - Codegen XML->Java
  - Skeleton
Adding a New Transport

- Adding a new transport framework is just a matter of
  - Creating a TransportSender
  - Creating a TransportReceiver
  - Registering them in axis2.xml
Reliable Messaging

- Invoking a service in a reliable manner
- Configuring a service
  - Adding control operation
- Deploying Sandesha on the client and server side
- Configuring the Client side
WS Security

- Invoking a service in a secure manner
  - Encrypt
  - Sign
  - Sign and Encrypt
- Deploying Rampart on the server side and client side
MTOM

- What is MTOM?
- What is the disadvantage with base64?
  - 8->6->8
- Advantage of using MTOM
  - Can send binary as it is
  - As base64
REST or POX

- Generating WSDL 1.1 and WSDL 2.0 rest binding
- Invoking a service the REST way
- No additional configuration required, same service can be exposed as
  - SOAP
  - or
  - REST
Questions????
Resources

- Latest resources can be found at
  - www.wso2.org
- Axis2 official web site
  - http://ws.apache.org/axis2
- Presentation material
  - http://people.apache.org/~deepal/colorado