LOG IN OR JOIN HOME REFCARDZ MICROZONES ZONES LIBRARY SNIPPETS TUTORIALS

Creating SOAP Messag Search

DZone

ENTERPRISE INTEGRATION

Meera Subbarao

Website

Creating SOAP Message Handlers in 3 Simple Steps - Part 1

08.13.2008 Email Tweet 0 Tweet 0 3 Share Views: 30014

The Enterprise Integration Zone is presented by DZone and FuseSource and JNBridge. For open source systems based on Apache Camel, ActiveMQ, or ServiceMix, look into FuseSource's training and technology. For .NET and Java interoperability, JNBridge has the answers.

This tutorial takes a look at SOAP Message handlers and how easy it is to write handlers using JAX-WS 2.0. JAX-WS 2.0 allows both regular Java classes and stateless EJBs(Session beans) to be exposed as web services. The JAX-WS 2.0 is the core specification that defines the web services standard for Java EE 5 specification. JAX-WS 2.0 is an extension of the Java API for XML-RPC (JAX-RPC) 1.0.

SOAP message handlers are used to intercept the SOAP message as they make their way from the client to the end-point service and vice-versa. These handlers intercept the SOAP message for both the request and response of the Web Service. If you are familiar with EJB interceptors, handlers are similar to EJB interceptors and are defined in an XML file.

A few typical scenarios where you would be using SOAP Message handlers are: to encrypt and decrypt messages, to support logging, caching and in some cases auditing, and in rare cases to provide transaction management as well.

So much for the theory. Lets see the three basic steps to use a simple log handler to intercept and print our SOAP messages (request and response).

In this tutorial, we are going to expose an **EJB 3 stateless session bean** as a web service which is simple and can be done by adding the **@WebService** annotation. So, here comes the source code for the interface as well as the implementation class which is self explanatory:

Listing 1: Remote Interface

```
package com.ws;

import javax.ejb.Remote;

/**
 *
 * @author meerasubbarao
 */
@Remote
public interface HelloWebServiceRemote {

    String sayHello(String name);

}
```

Listing 2: Bean Implementation

Got a story? Tell us!

Recommended Links

Creating .NET-based Mappers and Reducers for Hadoop with JNBridgePro

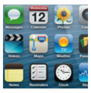
Design Pattern Flashcards for Integration

Integration Resources

Open Source Integration Tool Downloads

Why Open Source is Winning...

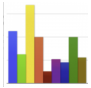
Spotlight Features



New iPhone 5 and iOS 6 Features for HTML 5 Developers

Take the HTML5 Poll, Win a DZone Swag Pack!

Juggling Multiple Versions of Java on OS X Mountain Lion



Weekly Poll: Scala, Friend or Foe? What's DZone's Consensus?

Popular at DZone

Embedded Highlights at JavaOne and Java Embedded @ JavaOne

DAO Layer - Generics to the Rescue

```
package com.ws;

import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 *
 * @author meerasubbarao
 */
@WebService
@Stateless
public class HelloWebServiceBean implements HelloWebServiceRemote {

   @WebMethod(operationName = "sayHello")
   public String sayHello(@WebParam(name = "name") String name) {
      return "Hello " + name;
   }


}
```

Road to JAX - A Chat with JClarity's Martijn Verburg and Kirk Pepperdine

What is lazy loading in Hibernate?

5 Tips to becoming a better programmer... Tip #1

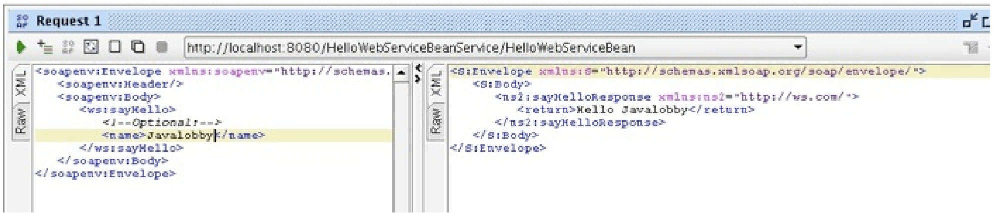
How to include a JSP Page in Servlet

Puppet Module for Maven Repository Artifact Downloads

See more popular at DZone

Subscribe to the RSS feed

You can package the two source files in a jar, deploy to your application server, and test it. Quite simple, right? For this tutorial, I am using **GlassFish V2** application server and **SoapUI** to test my web services. Shown below are the request and response from SoapUI:



http://localhost:8080/HelloWebServiceBeanService/HelloWebServiceBean

```
<soapenv:Envelope xmlns:soapenv="http://schemas.
   <soapenv:Header/>
   <soapenv:Body>
      <ws:sayHello>
         <!--Optional:-->
         <name>Javalobby</name>
      </ws:sayHello>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:sayHelloResponse xmlns:ns2="http://ws.com/">
         <return>Hello Javalobby</return>
      </ns2:sayHelloResponse>
   </S:Body>
</S:Envelope>
```

Now that we know our web services work, lets start writing the message handler, which as I said earlier is just 3 steps. So, what are these SOAP message handlers? They are simple Java classes that can used to modify SOAP messages; both request as well as response. These handlers have access to both the SOAP header as well as the body of the message.

Lets move on to create SOAP message handlers:

Step 1. Implement the SOAPHandler interface.

```
package com.ws;

import java.io.IOException;
import java.util.Collections;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

/**
 *
 * @author meerasubbarao
 */
public class LogMessageHandler implements SOAPHandler<SOAPMessageContext> {

   public boolean handleMessage(SOAPMessageContext messageContext) {
       return true;
   }

   public Set<QName> getHeaders() {
      return Collections.EMPTY_SET;
   }

   public boolean handleFault(SOAPMessageContext messageContext) {
      return true;
   }

   public void close(MessageContext context) {
   }

}
```

The **handleMessage()** method is invoked for both incoming as well as outgoing messages. Lets add a new method called **log()** and invoke this method from the handleMessage method. Shown below are both the methods:

```
01. private void log(SOAPMessageContext messageContext) {
02.        SOAPMessage msg = messageContext.getMessage(); //Line 1
03.     try {
04.        msg.writeTo(System.out);  //Line 3
05.     } catch (SOAPException ex) {
06.        Logger.getLogger(LogMessageHandler.class.getName()).log(Level.SEVERE
              null, ex);
07.     } catch (IOException ex) {
08.        Logger.getLogger(LogMessageHandler.class.getName()).log(Level.SEVERE
              null, ex);
09.     }
10. }
```

In line 1, we are retrieving the SOAPMessage from the message context. Line 3 will print the incoming and outgoing messages in our GlassFish console.

Invoke this method from within the **handleMessage**() as shown:

```
public boolean handleMessage(SOAPMessageContext messageContext)
      log(messageContext);
      return true;
   }
```

Step 2: Create the XML file for the Handler Chain.

Create this XML file in the same package as the web service with the name LogMessage_handler.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-name>com.ws.LogMessageHandler</handler-name>
      <handler-class>com.ws.LogMessageHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

Lets take a close look at the various elements used above:

1. **<handler-chains>** is the root element that will contain a list of all handler chains that are defined for the Web Service.
2. The **<handler-chain>** child element of the **<handler-chains>** element lists all the handlers in the handler chain.
3. Within the **<handler-chain>** element is defined the **<handler>**, each handler element must in turn specify the name and also the fully qualified name of the Java class that implements the handler. If you have more than one handler, specify each one of them within the handler-chain element.

Step 3: Invoking the Handler

The **@HandlerChain** annotation is used to define a set of handlers that are invoked in response to a SOAP message. So, within our **HelloWebServiceBean** implementaion, you need to make a simple change to invoke the Log Handler as shown below in Line 1:

```
01. package com.ws;
02.
03. import javax.ejb.Stateless;
04. import javax.jws.HandlerChain;
05. import javax.jws.WebMethod;
06. import javax.jws.WebParam;
07. import javax.jws.WebService;
08.
09. /**
10.  *
11.  * @author meerasubbarao
12.  */
13. @WebService
14. @Stateless
15. @HandlerChain(file = "LogMessage_handler.xml") // Line 1
16. public class HelloWebServiceBean implements HelloWebServiceRemote {
17.
18.    @WebMethod(operationName = "sayHello")
19.    public String sayHello(@WebParam(name = "name") String name) {
20.       return "Hello " + name;
21.    }
22. }
```

We added the annotation for the handlers; lets recompile, repackage and redeploy the application.

Now invoke the web service from **SoapUI** and see if it works. If it does, we should be able to see the request and response logged in our GlassFish console window.

Here is the final output:

```
**RemoteBusinessJndiName: com.ws.CustomerManagerRemote; remoteBusIntf:
com.ws.CustomerManagerRemote
LDR5010: All ejb(s) of [EJBWebServices] loaded successfully!
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws.com/"><soapenv:Header/><soapenv:Body><ws:sayHello>

<name>Javalobby</name>
</ws:sayHello></soapenv:Body></soapenv:Envelope>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body>
<ns2:sayHelloResponse xmlns:ns2="http://ws.com/"><return>Hello Javalobby</return>
</ns2:sayHelloResponse></S:Body></S:Envelope>
```

In this article, we saw how simple and easy it was to create and use SOAP Handlers to intercept request and response of SOAP messages. We implemented the SOAPHandler interface, wrote minimal XML to define the handler chain, and finally added one simple annotation to the web service implementation class. We were also able to test these web service using SoapUI.

In Part 2 of this series, we will see how to actually parse the SOAP Headers, and also use multiple handlers. Stay tuned for more..

Published at DZone with permission of its author, Meera Subbarao.

(Note: Opinions expressed in this article and its replies are the opinions of their respective authors and not those of DZone, Inc.)

Tags: Frameworks

AROUND THE DZONE NETWORK

JAVALOBBY
Using ZeroMQ Devices to Support Complex Network To...

ARCHITECTS
Structural Abstractions in Brains and Graphs

JAVALOBBY
Oracle Public Cloud Java Service Development Tools...

SERVER
DevOps Cloud Patterns

ARCHITECTS
Improving Site Performance with YSlow

MOBILE
I think I just snapped

YOU MIGHT ALSO LIKE

Introducing the New Date and Time API for JDK 8

New iPhone 5 and iOS 6 Features for HTML 5 Developers

Weekly Poll: Scala, Friend or Foe? What's DZone's Consensus?

Is Java Dead or Invincible?

HTML5 Web Storage – Cookies Are So 1994!

Throw Checked Exceptions Like Runtime Exceptions in Java

What Web Developers Need to Know about IE10 Compatibility

Designers vs. Engineers

Choosing Static vs. Dynamic Languages for Your Startup

How Much Longer Will Computer Science Departments Last?

NoSQL-like Approaches to PostgreSQL, a reply to Martin Fowler

Hidden Code

Weekly Poll: Where Do Developers Hang Out Online?

Don't Do it Wrong: Put that Puppet in a Box and Use Nexus for Devops

Getting Started with Twitter Bootstrap

POPULAR ON SNIPPETS

- Spring Batch - Hello World
- Is Hibernate the best choice?
- 9 Programming Languages To Watch In 2011
- Lucene's FuzzyQuery is 100 times faster in 4.0
- How to Create Visual Applications in Java?
- Introduction to Oracle's ADF Faces Rich Client Framework
- Time Slider: OpenSolaris 2008.11 Killer Feature
- Interview: John De Goes Introduces a Newly Free Source Code Editor

LATEST ARTICLES

- W3C Accepts Microsoft's Multiple Pointer Types Submission
- Game Development 101, Part 2
- Software Consulting, There are Apps for That!
- Redis as Memcached Replacement?
- Seamlessly Static Meta Model generation
- CloudStack 4.0 Release Schedule Update
- Reading and Text Mining a PDF-File in R
- Entity Framework 5 First Steps – Mapping

Search

DZone

Refcardz
Tech Library
Snippets
About DZone
Tools & Buttons

Book Reviews
IT Questions
My Profile
Advertise
Send Feedback

Topics

HTML5
Cloud
.NET
PHP
Performance
Agile

Windows Phone
Mobile
Java
Eclipse
Big Data
DevOps

Follow Us

Google +
Facebook
LinkedIn
Twitter

Controlling complexity is the essence of computer programming.

— Brian Kernigan