Service Routing

Service Routing

One thing you'll often need to do when developing services, is to develop a new version while keeping the old version running. This guide shows how to develop a simple service router that will scan the incoming message then direct the message to the appropriate service version.

For more complex requirements you might also wish to check out Apache Camel for full support for the Enterprise Integration Patterns with CXF

One common practice to version web services is using XML namespaces to clearly delineate the versions of a document that are compatible. For example:

```
<wsdl:types>
   <schema
      targetNamespace="http://apache.org/2007/03/21/hello_world_xml_http/mixed/types"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="sayHi">
         <complexType />
      </element>
      ..........
   </schema>
</wsdl:types>
```

Among many different possible implementations of service routing, one simple way ("simple" in terms of amount of code you have to write, but it does require a certain extent of familiarity with CXF internal architecture) to do this is by writing a CXF interceptor that acts as a routing mediator.

Firstly we need to have a dummy service with an intermediary interceptor registered. This intermediary interceptor sits at the very beginning of the interceptor chain, this is to make sure the intermediary interceptor is the first interceptor being invoked in the message pipeline. The intermediary interceptor scans the incoming message for example, the schema namespace, then directs the message to the desired endpoint according to user programmed strategy.

Lets see the code:

Example 1: The server - this server has three endpoints: one endpoint for the dummy service, another two endpoints are different versions of Greeter service

```
import javax.xml.ws.Endpoint;

import org.apache.cxf.jaxws.EndpointImpl;
import org.apache.cxf.testutil.common.AbstractBusTestServerBase;
import org.apache.hello_world_mixedstyle.GreeterImplMixedStyle;


public class Server extends AbstractBusTestServerBase {

    protected void run() {
        //implementor1 and implementor2 are published using local transport
        Object implementor1 = new GreeterImplMixedStyle();
        String address1 = "local://SoapContext/version1/SoapPort";
        Endpoint.publish(address1, implementor1);

        Object implementor2 = new GreeterImplMixedStyle();
        String address2 = "local://SoapContext/version2/SoapPort";
        Endpoint.publish(address2, implementor2);

        //A dummy service that acts as a routing mediator
        Object implementor = new GreeterImplMixedStyle();
        String address = "http://localhost:9027/SoapContext/SoapPort";
        javax.xml.ws.Endpoint jaxwsEndpoint = Endpoint.publish(address, implementor);

        //Register a MediatorInInterceptor on this dummy service
        EndpointImpl jaxwsEndpointImpl = (EndpointImpl)jaxwsEndpoint;
        org.apache.cxf.endpoint.Server server = jaxwsEndpointImpl.getServer();
        org.apache.cxf.endpoint.Endpoint endpoint = server.getEndpoint();
        endpoint.getInInterceptors().add(new MediatorInInterceptor());
    }

    public static void main(String[] args) {
        try {
            Server s = new Server();
            s.run();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(-1);
        } finally {
            System.out.println("done!");
        }
    }
}
```

Example 2: The intermediary interceptor

```
import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

import org.apache.cxf.Bus;
import org.apache.cxf.binding.soap.SoapMessage;
import org.apache.cxf.binding.soap.SoapVersion;
import org.apache.cxf.binding.soap.SoapVersionFactory;
```

```
import org.apache.cxf.bus.CXFBusFactory;
import org.apache.cxf.endpoint.Server;
import org.apache.cxf.endpoint.ServerRegistry;
import org.apache.cxf.interceptor.InterceptorChain;
import org.apache.cxf.interceptor.StaxInInterceptor;
import org.apache.cxf.message.Message;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.cxf.staxutils.DepthXMLStreamReader;
import org.apache.cxf.staxutils.StaxUtils;
import org.apache.cxf.transport.MessageObserver;


public class MediatorInInterceptor extends AbstractPhaseInterceptor<SoapMessage> {

    public MediatorInInterceptor() {
        super();
        setPhase(Phase.POST_STREAM);
        addBefore(StaxInInterceptor.class.getName());
    }

    public void handleMessage(SoapMessage message) {
        String schemaNamespace = "";
        InterceptorChain chain = message.getInterceptorChain();

        //scan the incoming message for its schema namespace
        try {
            //create a buffered stream so that we get back the original stream after scaning
            InputStream is = message.getContent(InputStream.class);
            BufferedInputStream pis = new BufferedInputStream(is);
            pis.mark(pis.available());
            message.setContent(InputStream.class, pis);

            //TODO: process attachements

            String encoding = (String)message.get(Message.ENCODING);
            XMLStreamReader reader = XMLInputFactory.newInstance().createXMLStreamReader(pis, encoding);
            DepthXMLStreamReader xmlReader = new DepthXMLStreamReader(reader);

            if (xmlReader.nextTag() == XMLStreamConstants.START_ELEMENT) {
                String ns = xmlReader.getNamespaceURI();
                SoapVersion soapVersion = SoapVersionFactory.getInstance().getSoapVersion(ns);
                //advance just past header
                StaxUtils.toNextTag(xmlReader, soapVersion.getBody());
                //past body.
                xmlReader.nextTag();
            }

            schemaNamespace = xmlReader.getName().getNamespaceURI();

            pis.reset();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }

        //Look up for all available endpoints registered on the bus
        Bus bus = CXFBusFactory.getDefaultBus();
        ServerRegistry serverRegistry = bus.getExtension(ServerRegistry.class);
        List<Server> servers = serverRegistry.getServers();

        //if the incoming message has a namespace contained "2007/03/21", we redirect the message
        //to the new version of service on endpoint "local://localhost:9027/SoapContext/version2/SoapPort"
        Server targetServer = null;
        for (Server server : servers) {
            targetServer = server;
            String address = server.getEndpoint().getEndpointInfo().getAddress();
            if (schemaNamespace.indexOf("2007/03/21") != -1) {
                if (address.indexOf("version2") != -1) {
                    break;
```

```
                }
            } else if (address.indexOf("version1") != -1) {
                break;
            }
        }

        //Redirect the request
        MessageObserver mo = targetServer.getMessageObserver();
        mo.onMessage(message);

        //Now the response has been put in the message, abort the chain
        chain.abort();
    }

}
```

A few things to note:

1. The `MediatorInInterceptor` is for SOAP binding, you can write a similar interceptor for XML binding etc.
2. We call `chain.abort` at the end of this interceptor to stop any further processing in the dummy service.
3. In this example, `implementor1` and `implementor2` are published using local transport. This is achieved by using an address like "local://SoapContext/version1/SoapPort". The `MediatorInInterceptor` looks up `ServerRegistry` to find endpoints hosted in the server then does the re-dispatch.
4. The `MediatorInInterceptor` is set to `POST_STREAM` phase and before `StaxInInterceptor`, this makes sure it is the very first interceptor to be invoked.