Table of Contents

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

Chapter 10. SOAP Headers

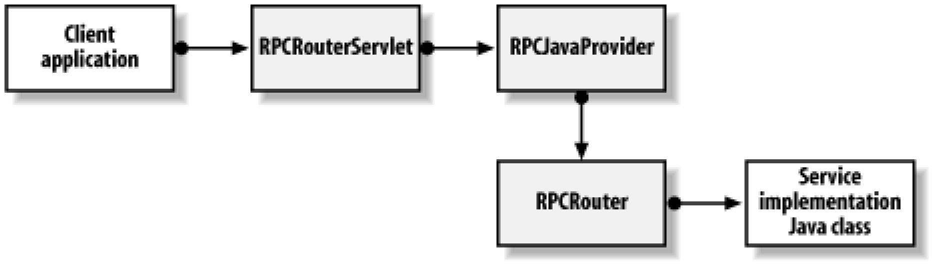
The SOAP envelope may contain a `Header` element that encompasses data outside the boundaries of an RPC (or other) style invocation. The header is an extension mechanism that can carry any kind of information that lies outside the semantics of the message in the body, but is nevertheless useful or even necessary for processing the message properly. Routing information might contain transaction identifiers, authentication keys, or other information related to the message's processing or routing.

Putting routing information in the SOAP header is useful when a message is sent to an intermediary, which in turn sends the message on to its final destination. In Chapter 9 we developed some proxy services that used another service to perform their tasks. In those cases, the client application was not aware that the service it was calling was actually a proxy. However, it's possible for proxy services to forward messages to specified endpoints or make use of other endpoints in the performance of their tasks. We'll develop an example of this kind of proxy, or intermediary, service.

10.1. Apache SOAP Providers and Routers

The Apache SOAP engine does not provide direct support for working with SOAP headers. However, it provides just enough Java class support and functional hooks to let you add headers without too much pain. Most of the work involved will be on the service side. Basically, we want a way to gain access to the SOAP header from within the service methods we implement. But before we delve into any code, let's look at the way Apache SOAP handles and routes SOAP messages. Figure 10-1 shows the path taken by an RPC message sent to an Apache SOAP service. The objects shown in gray are part of the Apache SOAP framework, while those in white are the Java classes we've been writing throughout the book.

Figure 10-1. The path from the client to the service implementation



10.2. Replacing the Provider and Router Classes

In order to make the SOAP header available to our service methods, we'll need to replace the router class with one of our own design. We could modify the `org.apache.soap.server.RPCRouter` class and rebuild the Apache SOAP source tree, but that's a bit messy; we'd have to go through the process again for every new release of the Apache SOAP engine. Luckily, Apache SOAP makes use of a concept called *pluggable providers*. This means that you can plug in your own provider to be used with a given service deployment. To use your own provider, you must implement your own provider class. This is just the hook we're looking for. Creating our own provider lets us use our own router, which can pass the SOAP header to our service methods. For instance, if we implement a provider class called `BetterRPCJavaProvider`, we can associate it with a service deployment by modifying the `isd:provider` element in the relevant deployment descriptor:

```
<isd:provider type="javasoap.book.ch10.services.BetterRPCJavaProvider"
```

10.3. An Apache SOAP Service That Handles SOAP Headers

We now have a provider and router that can pass the SOAP header to our service methods, so let's go ahead and implement a service that uses the SOAP header. The `urn:ProxyQuoteService` from Chapter 9 is a good place to start. Its implementation hardcoded the physical address of the back-end service as *http://mindstrm.com:8004/glue/urn:CorpDataServices*; we'll ask the client application to provide that information as part of the SOAP header. Our proxy service defaults to a bogus address of *http://mindstrm.com:8899/glue/urn:CorpDataServices* if no routing data appears in the header. You might use a similar technique if the service accesses a default back-end service but can also be directed to a client-specified service via the SOAP header.

Let's design our SOAP header so that it contains a single child element named `targetAddress`. This element contains the back-end service address that the proxy should use. The elements within the header may use `the SOAP-ENV:mustUnderstand` attribute with a value of `"1"` to indicate that the recipient of the envelope must understand the header element and properly use the data provided. The SOAP header sent by a client that demands the understanding of the `targetAddress` should look like this: