Java Web Services Using JAX-RPC

Student Workbook

itcourseware
Training Materials For IT Professionals

JAVA WEB SERVICES USING JAX-RPC

David Byrden

Published by ITCourseware, LLC., 7245 South Havana Street, Suite 100, Centennial, CO 80112

Contributing Authors: Julie Johnson and Jamie Romero

Editor: Jan Waleri

Special thanks to: Many instructors whose ideas and careful review have contributed to the quality of this workbook and the many students who have offered comments, suggestions, criticisms, and insights.

Copyright © 2011 by ITCourseware, LLC. All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photo-copying, recording, or by an information storage retrieval system, without permission in writing from the publisher. Inquiries should be addressed to ITCourseware, LLC., 7245 South Havana Street, Suite 100, Centennial, Colorado, 80112. (303) 302-5280.

All brand names, product names, trademarks, and registered trademarks are the property of their respective owners.

Contents

 © 2011 ITCourseware, LLC

 © 2011 ITCourseware, LLC

CHAPTER 1 - COURSE INTRODUCTION

COURSE OBJECTIVES

- Describe web services and how they are implemented with Java technologies.
- Design web service applications to meet the needs of new and legacy systems and requirements.
- Use SOAP to access web services.
- Describe web services with WSDL.
- Access middle-tier and back-end applications through web services.

 © 2011 ITCourseware, LLC

COURSE OVERVIEW

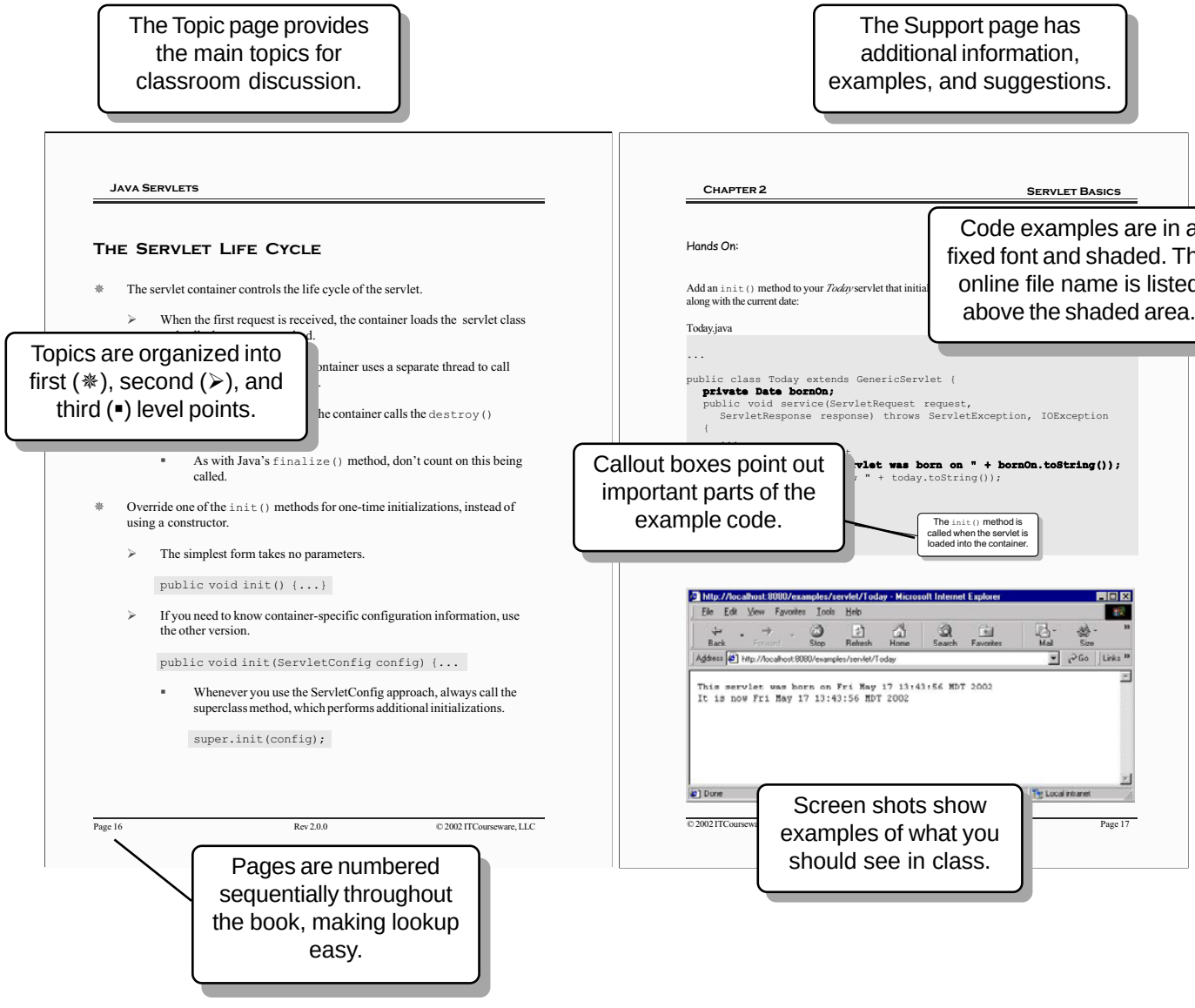
- **Audience**: Experienced XML and Java developers who need to develop and publish web services applications on the Internet.
- **Prerequisites**: *Fundamentals of XML* and *Java Programming*
- **Classroom Environment:**
  - A workstation per student.

 © 2011 ITCourseware, LLC

USING THE WORKBOOK

This workbook design is based on a page-pair, consisting of a Topic page and a Support page. When you lay the workbook open flat, the Topic page is on the left and the Support page is on the right. The Topic page contains the points to be discussed in class. The Support page has code examples, diagrams, screen shots and additional information. Hands On sections provide opportunities for practical application of key concepts. Try It and Investigate sections help direct individual discovery.

In addition, there is an index for quick lookup. Printed lab solutions are in the back of the book, as well as online if you need a little help.



SUGGESTED REFERENCES

Monson-Haefel, Richard. 2003. *J2EE Web Services*. Addison-Wesley, Boston, MA. ISBN 0321146182

Monson-Haefel, Richard and Bill Burke. 2006. *Enterprise JavaBeans, Fifth Edition*. O'Reilly & Associates, Sebastopol, CA. ISBN 059600978X

Nagappan, Ramesh, Robert Skoczylas, Rima Patel Sriganesh. 2002. *Developing Java Web Services*. John Wiley & Sons, Inc., New York, NY. ISBN 0471236403

Newcomer, Eric. 2002. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, Boston, MA. ISBN 0201750813

Singh, Inderjeet, Sean Brydon, Greg Murray, Vijay Ramachandran, Thierry Violleau, and Beth Stearns. 2004. *Designing Web Services with the J2EE™ 1.4 Platform: JAX-RPC, SOAP, and XML Technologies*. Addison-Wesley, Boston, MA. ISBN 0321205219

Topley, Kim. 2003. *Java Web Services in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA. ISBN 0596003994

http://java.sun.com/xml
http://webservices.xml.com
http://www.ws-i.org
https://jax-rpc.dev.java.net

 © 2011 ITCourseware, LLC

 © 2011 ITCourseware, LLC

CHAPTER 2 - INTRODUCTION TO WEB SERVICES

OBJECTIVES

- Explain the need for web services.
- Describe the web services application model.
- Define the role of web services in enterprise system architectures.

What is a Web Service?

- A *web service* is a set of functions, packaged as a component, that can be invoked over a network.
- Web services are based on open, freely available, non-proprietary standards.
- Web services implement the Service-Oriented Architecture (SOA); they are not object-oriented.
  - SOA is stateless, loosely-coupled, and message-based.
- Web services are not a part of the World Wide Web.
  - They can, and usually do, use the web's HTTP protocol.

 © 2011 ITCourseware, LLC

E-businesses take full advantage of the Internet and other networks in order to conduct commerce. Web services were invented to enable and enhance e-business. It is potentially possible for all formal business communications (orders, invoices, contracts, etc.) to occur electronically and in real time. If a website can publish text to a global audience, a business should be able to publish a service to other businesses worldwide, and take the first orders only moments later.

The dominant Internet applications today are email and the World Wide Web. Email has significant security problems that make it unsuitable for e-business. Many commercial applications have been built on the web, but in general they are Business-To-Consumer (B2C), rather than Business-to-Business (B2B). The trend in web development has always been to present data to humans for processing, but humans are too slow and unreliable to be an integral part of an architecture for e-business.

E-business has already been conducted for many years with Electronic Data Interchange (EDI). However, EDI is difficult to understand, has considerable implementation overheads, and requires careful coordination between client and server. Custom programming is often required for individual business collaborations. EDI is too inflexible to realize the e-business vision.

Distributed object frameworks, such as Java RMI, are not suitable for e-business, because they impose technological dependencies, overheads, or both. For example, RMI cannot be used unless both the client and server are Java-based.

Web services were invented, largely by Microsoft and IBM, as the foundation for e-business. They were rapidly adopted by major rivals in the industry. Although many specifications critical to secure, reliable e-business are newly-completed or still in development, the basic web service standards are already in widespread use.

SERVICE-ORIENTED ARCHITECTURE

- Web services are an implementation of a concept called Service-Oriented Architecture (SOA).

- SOA is useful where independent software components are connected by a network.

- The problems specific to this environment are:

  - Arbitrarily long network delays
  - Components going offline temporarily
  - No central control
  - Creation and removal of components at any time

- The distinguishing feature of SOA is that components are loosely-coupled and can be dynamically found when required.

- There are three roles in the SOA architecture: service provider, service consumer, and registry.

- There are three software entities in the SOA architecture: contract, proxy, and lease.

- Web services implement only four of these six concepts.

  - The service provider is a SOAP server.

  - The service consumer is a SOAP client.

  - UDDI provides registry services.

  - WSDL defines the contract, or interface of a web service.

 © 2011 ITCourseware, LLC

In SOA, the term *service* means a software module that may be invoked over the network. A service is controlled by a *service provider*. The provider may remove the service at any time (when it is not executing a request).

The users of services, called *service consumers*, cannot rely on a specific service always being there when required. Instead, they choose at runtime from multiple services (perhaps owned by competing providers).

The ability to choose a service dynamically has implications for B2B. The consumer may choose between the available services based on cost or other criteria. This creates a *service marketplace*.

To enable the dynamic substitution of one service for another, SOA strictly separates interfaces from implementations. A service's interface, called its *contract*, says nothing about how the service is implemented or what is its network address. Therefore, a single contract can be implemented by multiple services and providers.

To enable consumers to find services dynamically, SOA introduces the concept of a *registry*. This is a distributed repository of information about services. The information is placed there by service providers who wish their services to be used. In particular, the registry contains contracts. Consumers can ask about all the services that fulfill a specific contract. The registry will return the information necessary to use each service, such as its network address.

All of the above features of SOA are implemented in the web service framework. Two other SOA concepts are not implemented at present: leases and proxies.

A *lease* represents the right of a consumer to use a service for a specified period of time. Leases expire, allowing service providers to withdraw or alter their services without having to warn the consumers. Without leases, we can expect web service consumers to fail when this happens.

A *proxy* is a software component that executes in the consumer's computer and represents a remote service. It encapsulates network communications with the service, which is extremely useful. In SOA, proxies are supplied by service providers via the registry. Web service clients have to generate their own proxies.

DISTRIBUTED APPLICATIONS

- A *distributed application* is one whose parts are connected by a network.
- Distribution brings new design problems and failure modes to an application, such as network delays and partial failures.
- Web services may be invoked over slow, unreliable networks — specifically the Internet.
  - One of the services participating in an interaction may fail or go offline, while others continue.
  - It may be impossible to know which operations it completed before the failure.
  - Messages can become lost, and message sequences can arrive in the wrong order.
- Network message transport is always significantly slower than method calls within a process.
- SOA compensates for this by promoting simple interactions and complex, infrequent messages.

 © 2011 ITCourseware, LLC

The client of a web service is not always the ultimate user of the service. Web services are components that will be chained to form applications. For example, an online store might use a third-party web service to provide it with currency exchange rates, or to interpret localized postal addresses. A manufacturer might take an order for goods and, as part of that interaction, place orders with its suppliers for the necessary parts.

Web service applications must therefore be designed to cope with network failures. SOA provides useful guidelines:

- Services should communicate by message passing. This allows the use of message queues, transactions, and other techniques that compensate for network failures.
- Messages should be large and infrequent. This reduces the number of communications and communication failures.
- Services should be designed and deployed without consideration of the clients that will use them.
- Services should be stateless. That is, they should not hold conversational state. This allows simpler service implementations.
- Services and clients should only share contracts (promises of service behavior) and schemas (structure of messages).

XML and Java

- The specifications for web services leave many options open, but they insist on XML for data and messages.
  - They also insist on W3C XML Schema as the XML schema language.
- XML supports the e-business concept in several ways:
  - XML is completely platform-neutral.
  - XML support is already integrated into the major platforms.
  - XML knowledge is widespread among developers.
  - XML supports Unicode and, therefore, supports internationalization.
  - XML can be edited by humans, which aids development and testing.
  - Editors and other useful tools for XML are commonplace.
- Java is an ideal platform for building a web service, because of its portability.
  - The software can easily be moved to a different server platform or hosting provider.
- XML capabilities have been integrated into the Java platform libraries.
- Java's J2EE provides the infrastructure to host enterprise-level web services.

 © 2011 ITCourseware, LLC

Java has several APIs, new and old, that support web services.

- The core Java APIs have had basic XML support for some time.
- The Java Web Services Developer Pack (JWSDP) adds APIs specific to web services.
- Release 1.4 of J2EE includes these APIs as standard.

JAXP (Java API For XML Processing) is a collection of Java libraries for XML processing:

- SAX — A low-overhead parser
- DOM — An XML object model
- XSLT — A means to translate between XML document formats

The Java web service APIs are:

- SAAJ — For editing SOAP messages and working with attachments
- JAX-RPC — For building J2EE web services
- JAXR — For using web service directories
- JAX-WS — For Java EE 5 web services

Enterprise Java Beans (EJB) are software components used to build robust, high-performance server applications.

- EJBs execute in a J2EE container, which manages them and provides an advanced infrastructure.
- It is intended that enterprise-level web services will be hosted in the J2EE platform.
- Lightweight web services can be hosted in the Java Web Container.

For more on the W3C XML Schema Language: ***http://www.w3.org/XML/Schema/***

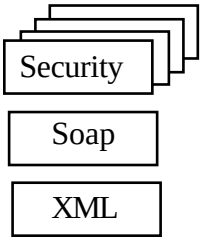
WEB SERVICES STRUCTURE

- The majority of web service standards are published by the World Wide Web Consortium (W3C).
  - The structure of the web service framework was outlined by the W3C in 2001.
- W3C has described three activities that are performed in the framework:
  - Communication
  - Description
  - Discovery
- Each of these activities is performed by a software stack.
  - A *stack* is a chain of software modules, called *layers*, each dependent on the one beneath.
  - Each layer has an open specification and can be implemented by any vendor.
  - The lowest, simplest layers of the stacks were defined first.

 © 2011 ITCourseware, LLC

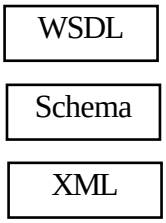
The *communications stack*, or *wire stack*, is the first of the three web service stacks. It defines the messages that invoke web services. It does not define a transport protocol for those messages.

The lowest layer of the communications stack is XML itself. The format of the messages is defined in the next layer, by a specification called the Simple Object Access Protocol (SOAP). It is layered on top of XML, because SOAP messages are XML documents.



The layers above SOAP are still less mainstream. There are specifications for security, reliable messaging, binary attachments, and other features. These are independent, and they form a single layer of the stack. In some cases, there are rival specifications for the same feature.

The *description stack* defines the contracts that web services will publish to describe themselves to their clients. Contracts are a feature of the SOA architecture.



This stack begins with XML, followed by general-purpose XML schema languages. These languages can describe the format of XML documents. Next is the WSDL language, which specifically describes web service interfaces at the API level. The higher layers of this stack allow a service to define the sequence of messages or API calls that comprise a business interaction.

The *discovery stack* defines the registries in which clients find services. This stack contains only one item: UDDI, the specification of a distributed directory.

The current versions of the web service specifications are: SOAP 1.2, WSDL 1.1, and UDDI 3.0.

The W3C is responsible for HTML and XML; ***http://www.w3.org/***. SOAP is the messaging protocol of web services. Its specification can be found at ***http://www.w3.org/TR/soap***. The Web Services Description Language (WSDL) specification is at ***http://www.w3.org/TR/wsdl***. The UDDI specifications are maintained by OASIS at ***http://www.uddi.org/***.

Why Web Services?

- Web services are concerned only with making applications communicate.
  - They give applications a simple interface and optionally publicize them in directories.
- A web service interface can be retrofitted to an existing application at low cost.
- The benefits of web services are:
  - They are based on open standards.
  - They are supported by all the major software platforms, including J2EE and .NET.
  - They allow loose coupling, which lowers maintentance costs.
- Web services can integrate vertical applications within the enterprise.

 © 2011 ITCourseware, LLC

Web services are merely a form of communication for distributed applications. They have no capabilities that were not available in earlier frameworks. When compared to DCE, CORBA, DCOM, or RMI, they have a small feature list. So what are the reasons to use them?

Web services are independent of language and platform. A web services interface can be added to almost any application, whether object-based, procedural, or event-driven. The footprint of the software is so small that it is practical for mobile devices. Therefore, web services can be used to integrate legacy applications.

It is common for enterprises to have vertical applications that cannot communicate easily. Even when a framework, such as CORBA, is available, the cost of the programming may be prohibitive. This leads to unsatisfactory solutions, such as manual re-entry of data, batch copying, etc.

Web services offer a way to achieve this integration. They have significant tool support from vendors: IBM, Sun, Oracle, Microsoft, HP, and BEA all offer web service integration tools. Web service capabilities are also appearing in traditional integration tools. The rapid adoption of web services by the software industry is reminiscent of the World Wide Web.

Criticisms of Web Services

The web services architecture has been criticized for several reasons:

- The architecture is very basic, with no objects, references, or pointers.
- The binding of SOAP to HTTP breaks the HTTP specification by using the **POST** verb for carrying messages. **POST** is intended for editing resources.
- The HTTP binding cheats older firewalls, because RPC invocations are dangerous and should be controlled. The HTTP binding disguises them as harmless documents.
- SOAP has no support for sessions.
- XML is verbose.

RPC and Document Styles

- The messages that invoke web services are currently defined by the SOAP standard.
- They can have two styles: Document or RPC.
  - A *Document-style* message carries an XML element that may conform to any schema.
  - An *RPC-style* message invokes a specific procedure in the web service.
- RPC messages are synchronous; a reply message is sent as soon as the procedure completes.
- Document messages can be asynchronous; there may be a reply much later, or there may be no reply.
- The RPC style was initially the most popular, but now Document style dominates.

 © 2011 ITCourseware, LLC

The web service specifications, including the message specification SOAP, give the implementor many options and choices. One of these is the message style.

An RPC-style message is the wire format of a remote procedure call. SOAP was originally designed around these messages. An RPC message carries the procedure's input and output parameter values encoded as XML.

But RPC-style messaging leads to tight coupling between the client and service, for the following reasons:

- A client thread dispatches the message and then blocks until the reply arrives.
- Both client and server know that a specific behavior in the server is invoked.

Document-style messages conform to the SOA architecture. They are much simpler in structure than RPC, leading to a reduced software footprint. They promote loose coupling, for the following reasons:

- If implemented asynchronously, delays in the server do not delay the client's thread.
- The client and server may have different interpretations of the document they exchange.
- The client does not know what, if anything, the server will do with the document.

Asynchronous messaging opens up new possibilites for the application architecture, such as having messages copied to multiple subscribers. But it brings additional problems:

- Transactions are difficult to implement on top of asynchronous messages.
- An ordered sequence of asynchronous messages may arrive in incorrect order.
- System failures go unnoticed for much longer.
- Senders must be able to receive replies at any time, increasing their overhead.

WEB SERVICE INITIATIVES

- The basic web service specifications (SOAP, WSDL, UDDI) are not sufficient for commercial use.
- Several advanced specifications have been developed, in some cases competing with each other.
- The Web Services Interoperability Organization (WS-I) is a vendor initiative working outside of any standards body.
  - Its goal is to ensure that web service tools from different vendors will work together.
- SOAP 1.1, although widely used, was never approved by a standards body.
- The W3C have developed a new version, SOAP 1.2.

 © 2011 ITCourseware, LLC

The fundamental web service specifications (SOAP, WSDL, UDDI) have so many ambiguities and options that even fully-compliant web service tools may not be able to communicate. Microsoft, IBM, and other vendors founded WS-I to correct this situation. Organizations such as W3C were deemed too slow-acting, so WS-I remains independent. Over 160 vendors are now members.

WS-I has defined a Basic Profile (WSBasic), which is a template of a genuinely interoperable web service. It also provides supporting resources, such as a working sample application and interoperability tests.

The Web Services Interoperability Organization. ***http://www.ws-i.org/***.

SOAP 1.2: the first version of SOAP to be developed by a standards body. It removes the ambiguities of SOAP 1.1 and improves the HTTP binding.

LABS

❶ There is a website at ***http://www.xmethods.net/*** that maintains a list of public web services. Examine some of the services listed. Note that each one is marked as either RPC or Document style.

❷ Try some of the web services. Observe the XML request and response messages.

 © 2011 ITCourseware, LLC

 © 2011 ITCourseware, LLC

CHAPTER 6 - MESSAGE HANDLERS AND ATTACHMENTS

OBJECTIVES

- Use a JAX-RPC message handler to log SOAP messages.
- Describe the wire format of SOAP data.
- Edit SOAP messages directly with the SAAJ API.
- Add attachments to a web service with the SOAP with Attachments specification.

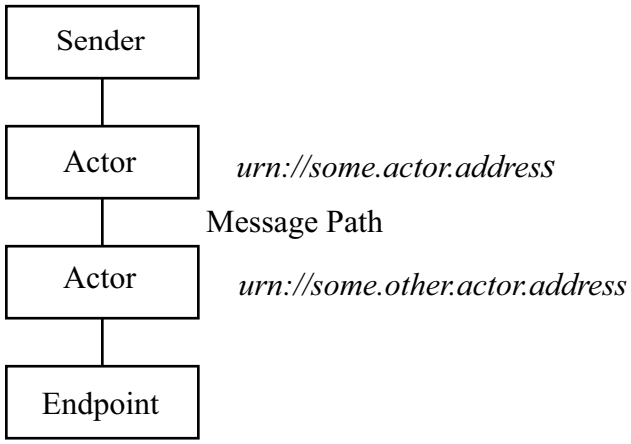
ACTORS AND SOAP HEADERS

- A SOAP message is annotated by adding SOAP headers to it.
  - The headers can contain arbitrary XML elements.
- Messages may pass through actors on the way to their endpoint.
  - An *actor* is a message filter, located at a network node, at the sender or at the endpoint.
  - The actors may add, read, and remove SOAP headers.
  - Actors are used as modules to extend the SOAP messaging model.

 © 2011 ITCourseware, LLC

The diagram shows a SOAP message path. Any number of actor nodes may be inserted between the sender of a message and the endpoint.

Actors are often paired, with the one actor adding headers to the messages and a later actor removing them. For example, the first actor could encrypt the message and add a header stating how it was done. The second actor could remove the header, read the information and decrypt the message. The sender and the endpoint would see no changes, but the central portion of the message path would now be more secure.



Every actor is required to have a unique URI. The URI is an identifier, not necessarily the same as the network address of the actor node.

An actor need not be a separate hardware or software module. To emphasize this, the term "actor" is replaced by "role" in the SOAP 1.2 specification.

The next message example has an annotation. Header entries are application-specific and should be in their own namespaces.

```
<SOAP:Envelope xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/' >
   <SOAP:Header>
      <APP:encrypted xmlns:APP='urn:some/applications/address'
         SOAP:actor='urn:some/actors/address' />
   </SOAP:Header>
   <SOAP:Body> <!-- Payload data elements go here...  -->
   </SOAP:Body>
</SOAP:Envelope>
```

A header entry should indicate the actor it is intended for, by using the **actor** attribute from the SOAP namespace. Its value is the identifying URI of the target actor.

In the JAX-RPC framework, actors can be conveniently implemented as message handler classes. Since handlers can be installed independently of applications, these actors are also reusable components.

MESSAGE HANDLERS

- JAX-RPC calls an actor a "message handler."
  - A *message handler* is an object that stands between the network and the JAX-RPC application.
  - Any number of handlers can be installed.
- Handlers do not need to have an effect on application code — they can be installed by the JAX-RPC configuration files.
  - They can also be installed programatically.
- A configuration file can install handlers by specifying them in a **<handlerChains>** element:

```
<handlerChains>
  <chain runAt="client"> <!-- specify "client" or "server" -->
    <handler  className="MyHandler1"/>
    <handler  className="MyHandler2"/>
  </chain>
</handlerChains>
```

- Messages to and from the application will pass through instances of all the handler classes listed.
  - Requests will encounter the handlers in the order specified.
  - Responses and faults will encounter the handlers in reverse order.

 © 2011 ITCourseware, LLC

volume-server/config-interface.xml

```
<?xml version="1.0"?>

<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
   <service name="VolumeCalc"
        targetNamespace="http://www.example.com/VolumeCalc"
        typeNamespace="http://www.example.com/VolumeCalcType"
        packageName="volume">
      <interface name="volume.VolumeCalcIF">
         <handlerChains>
            <chain runAt="server">
               <handler className="handler.Logger"/>
            </chain>
         </handlerChains>
      </interface>
   </service>
</configuration>
```

This installs a server-side handler chain.

rpc-volume-client/config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
   <wsdl location="http://localhost:8080/volume/volumecalc?wsdl"
         packageName="generated">
       <handlerChains>
          <chain runAt="client">
             <handler className="handler.Logger"/>
          </chain>
       </handlerChains>
   </wsdl>
</configuration>
```

This installs a client-side handler chain.

Implementing a Message Handler

- Your JAX-RPC handlers must implement the **javax.xml.rpc.handler.Handler** interface.
  - Use the **init()** and **destroy()** methods for initialization and cleanup, respectively.
  - Specify which SOAP headers this handler is processing with the **getHeaders()** method.
    - Return an empty array of **QName** objects if your handler does not work with SOAP headers.
  - Add your handler code to the **handleRequest()**, **handleResponse()**, and **handleFault()** methods.
- All three **handle*XXX*()** methods take a **MessageContext** object as a parameter.
  - You can store data in the **MessageContext** to be retrieved later by the message endpoint with the **setProperty()** method.
  - You can also cast the **MessageContext** to a **SOAPMessageContext** object to gain access to the **SOAPMessage** itself.
- For convenience, you can extend the **GenericHandler** class instead of implementing the **Handler** interface.
  - **GenericHandler** provides empty implementations for all of **Handler**'s methods except for **getHeaders()**.

 © 2011 ITCourseware, LLC

volume-server/Logger.java

```
...
public class Logger extends GenericHandler {

   public QName[] getHeaders() {
      return new QName[0];
   }

   public boolean handleRequest(MessageContext context) {
      System.out.println("**Start Message Request**");
      logMessage(context, System.out);
      System.out.println("**End Message Request**");
      return true;
   }

   public boolean handleResponse(MessageContext context) {
      System.out.println("**Start Message Request**");
      logMessage(context, System.out);
      System.out.println("**End Message Request**");
      return true;
   }

   private void logMessage(MessageContext context, OutputStream out) {
      try {
         SOAPMessageContext soapContext = (SOAPMessageContext)context;
         SOAPMessage message = soapContext.getMessage();
         message.writeTo(System.out);
      }
      catch (Exception e) {
         System.err.println(e);
      }
   }
}
```

Log the request and response messages to standard out.

Try It:

Run *buildanddeploy.bat* in this chapter's *volume-server* directory. Then run *build.bat* in the *rpc-volume-client* directory. Finally, run *VolumeClient.java:* **java VolumeClient**. The client-side **Logger** will output the request and response to the command window. The server-side **Logger** will write to the application server's log file.

Note:

Handlers must be stateless. That is, they must retain no memory of the messages they have already seen. This allows servers to pool large numbers of handler objects for a faster response.

SOAP ENCODING

- The data values carried in a SOAP message body are always encoded as XML.
  - However, there is more than one way to write a given value as XML.
- SOAP defines two approaches for writing data: *encoded* and *literal*.
- The content of a literal message's **Body** conforms to a schema.
  - The preferred language to use is W3C XML Schema.
- The content of an encoded message's **Body** is structured in some formal way.
  - As a benefit, standard tools may decode the data, rather than feeding it raw to the receiver.
  - As a disadvantage, it may not be possible to validate the encoded data with a schema.
- The SOAP specification uses *SOAP Encoding* as an example of an encoding style.
  - A large proportion of early web services use SOAP Encoding.
- JAX-RPC uses SOAP Encoding by default.
  - SOAP Encoding is especially popular with RPC-style web services.
  - Microsoft .NET web services use literal by default.
- SOAP Encoding prevents validation and is a source of other problems, and is, therefore, deprecated in the SOAP 1.2 specification.

 © 2011 ITCourseware, LLC

SOAP Encoding was invented early in the history of SOAP. When W3C XML Schema was too immature to encode arrays of data, SOAP Encoding extended it with a namespace ***http://schemas.xmlsoap.org/soap/encoding/*** containing an **Array** type. But XML Schema is now mature and no longer needs this enhancement.

Data values in SOAP-Encoded messages usually declare their types (perhaps a waste of bandwidth, because most clients and servers will already know the types). Here is an example:

```
<height xsi:type="xsd:double" >4.8</height>
```

As the example shows, the declared type can be taken from the XML Schema namespace, because SOAP Encoding extends XML Schema.

SOAP Encoding adds another feature called *multireference values*. It allows elements in the message body to be shared by reference, eliminating duplicates. Unfortunately, this makes schema validation generally impossible.

encoded-request.xml

```
<env:Envelope
     xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:ns0="http://www.example.com/VolumeCalc"
     xmlns:ns1="http://www.example.com/VolumeCalcType"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <env:Body>
      <ns0:getCylinderVolume>
         <Cylinder_1 href="#ID1" />
      </ns0:getCylinderVolume>

      <ns1:Cylinder id="ID1" xsi:type="ns1:Cylinder">
         <height xsi:type="xsd:double">4.8</height>
         <radius xsi:type="xsd:double">1.3</radius>
      </ns1:Cylinder>
   </env:Body>
</env:Envelope>
```

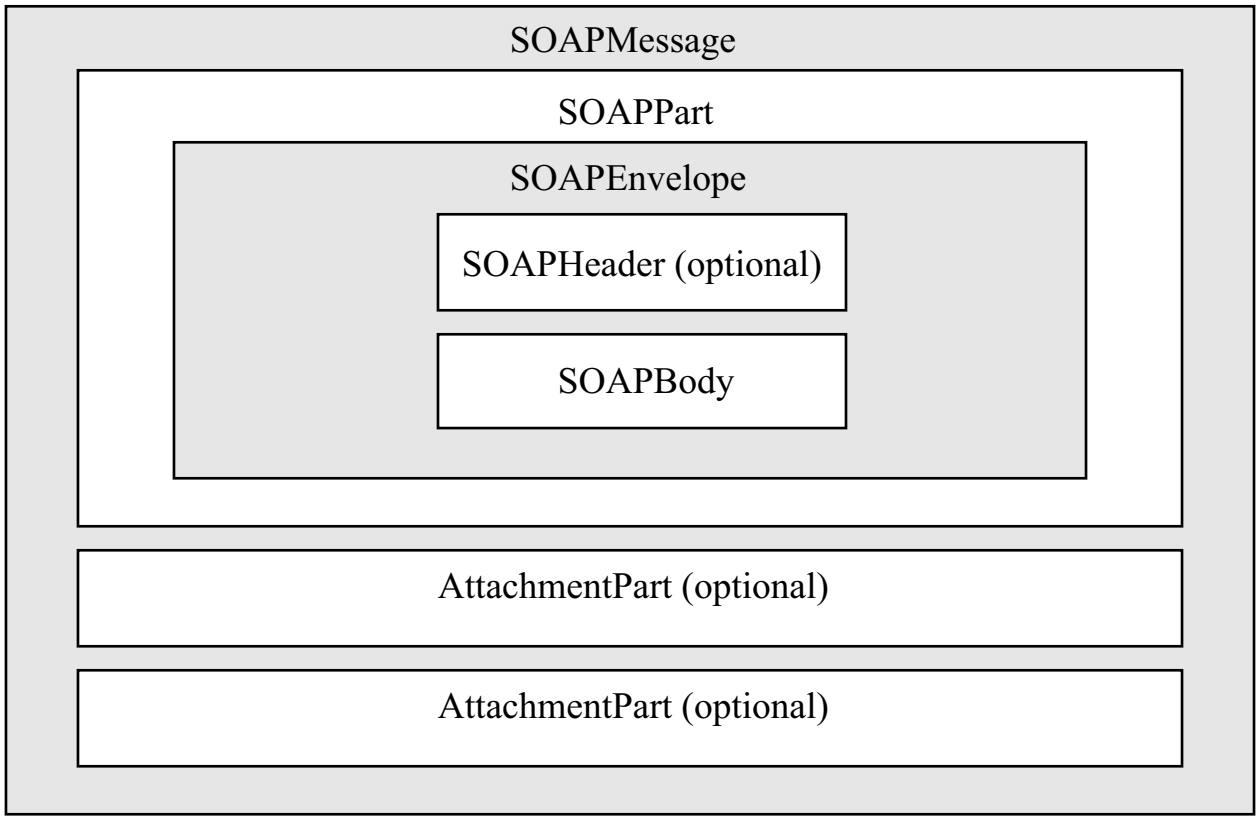
Note:

You can pass the **-fdocumentliteral** or **-frpcliteral** arguments to wscompile to override the default SOAP Encoding for interoperability purposes.

SAAJ

- The SOAP with Attachments API for Java (SAAJ) is a library for building and editing SOAP messages.
  - JAX-RPC message handlers see the messages in terms of the SAAJ API.
  - SAAJ is in the **javax.xml.soap** package.
- SAAJ is based on the DOM API, extending it with SOAP-specific features.
  - Its interfaces, such as **SOAPElement**, extend DOM interfaces, such as **Element**.
- Editing SOAP messages with SAAJ is easier than with DOM.
  - SOAP-specific methods, such as **getEnvelope()**, simplify the navigation.
- SAAJ was designed as a generic add-on for messaging APIs.
  - JAX-RPC is only one of many messaging APIs.

 © 2011 ITCourseware, LLC


SOAPMessage
SOAPPart
SOAPEnvelope
SOAPHeader (optional)
SOAPBody
AttachmentPart (optional)
AttachmentPart (optional)

IMPLEMENTING A SAAJ CLIENT

- You can build a web service client with SAAJ, instead of JAX-RPC, when you want more control over the SOAP request.
  - Create an empty **SOAPMessage** with the **MessageFactory** **createMessage()** method.

```
MessageFactory mFactory = MessageFactory.newInstance();
SOAPMessage message = mFactory.createMessage();
```

  - Retrieve the **SOAPPart**, **Envelope**, and **SOAPBody** from the **SOAPMessage**.
  - Add any necessary namespace declarations with the **SOAPEnvelope** **addNamespaceDeclaration()** method.

```
envelope.addNamespaceDeclaration("xsd",
   "http://www.w3.org/2001/XMLSchema");
```

  - Create a **Name** object to wrap the name and namespace of each element or attribute that you wish to add to the **SOAPBody**.

```
Name functionName = env.createName("functionName",
   "pre", "http://www.example.com/NamespaceName");
```

  - Use the **addBodyElement()**, **addChildElement()**, **addAttribute()**, or **addTextNode()** methods to add content to the **SOAPBody**.
    - Add immediate children to the **SOAPBody** with the **addBodyElement()** method.
    - Add more deeply-nested children with the **addChildElement()** method.

 © 2011 ITCourseware, LLC

saaj-volume-client/VolumeClient.java

```
import javax.xml.soap.*;

public class VolumeClient {
   public static void main(String[] args) throws Exception {
      String nsXSD = "http://www.w3.org/2001/XMLSchema";
      String nsXSI = "http://www.w3.org/2001/XMLSchema-instance";
      String nsTNS1 = "http://www.example.com/VolumeCalc";
      String nsTNS2 = "http://www.example.com/VolumeCalcType";

      MessageFactory mFactory = MessageFactory.newInstance();
      SOAPMessage message = mFactory.createMessage();
      SOAPPart sp = message.getSOAPPart();
      SOAPEnvelope env = sp.getEnvelope();
      SOAPBody body = env.getBody();

      env.addNamespaceDeclaration("xsd", nsXSD);
      env.addNamespaceDeclaration("xsi", nsXSI);
      env.addNamespaceDeclaration("tns1", nsTNS1);
      env.addNamespaceDeclaration("tns2", nsTNS2);
      env.setEncodingStyle("http://schemas.xmlsoap.org/soap/encoding/");

      SOAPBodyElement bodyElem1 = body.addBodyElement(
         env.createName("getCylinderVolume", "tns1", nsTNS1));

      SOAPElement subElem1 = bodyElem1.addChildElement("Cylinder_1");
      subElem1.addAttribute(env.createName("href"), "#ID1");

      Name xsiType = env.createName("type", "xsi", nsXSI);
      SOAPBodyElement bodyElem2 = body.addBodyElement(
         env.createName("Cylinder", "tns2", nsTNS2));
      bodyElem2.addAttribute(env.createName("id"), "ID1");
      bodyElem2.addAttribute(xsiType, "tns2:Cylinder");

      SOAPElement subElem2 = bodyElem2.addChildElement("height");
      subElem2.addAttribute(xsiType, "xsd:double");
      subElem2.addTextNode("4.8");

      SOAPElement subElem3 = bodyElem2.addChildElement("radius");
      subElem3.addAttribute(xsiType, "xsd:double");
      subElem3.addTextNode("1.3");
      ...
   }
}
```

Invoke the **getCylinderVolume** method.

Pass a **Cylinder** object.

Add the **height** and **radius**.

SENDING A SAAJ MESSAGE

- Send the **SOAPMessage** to a destination with the **SOAPConnection** class.
  - Retrieve an instance of **SOAPConnection** from the **SOAPConnectionFactory**.

```
SOAPConnectionFactory f = SOAPConnectionFactory.newInstance();
SOAPConnection conn = f.createConnection();
```

  - Pass the **SOAPMessage** object and the location of the web service to the blocking **call()** method.

```
SOAPMessage response = conn.call(soapMessage,
  "http://localhost:8080/context/endpoint");
```

  - The returned **SOAPMessage** represents the SOAP response from the web service that you can traverse to retrieve the return value.

 © 2011 ITCourseware, LLC

saaj-volume-client/VolumeClient.java

```
...
public class VolumeClient {
   public static void main(String[] args) throws Exception {
      ...
      System.out.println("SOAP REQUEST");
      System.out.println("****************************************");
      message.writeTo(System.out);
      System.out.println("****************************************\n");

      String endpoint = "http://localhost:8080/volume/volumecalc";
      SOAPConnectionFactory fac = SOAPConnectionFactory.newInstance();
      SOAPConnection conn = fac.createConnection();
      SOAPMessage response = conn.call(message, endpoint);
      conn.close();

      System.out.println("SOAP RESPONSE");
      System.out.println("****************************************");
      response.writeTo(System.out);
      System.out.println("****************************************\n");
   }
}
```

Try It:

Compile and run *VolumeClient.java* in the *saaj-volume-client* directory to invoke the volume calculator web service from a SAAJ client.

Note:

A **SOAPConnection** can only send RPC messges and may or may not be supported by a SAAJ implementation.

ATTACHMENTS

- It is sometimes necessary to send binary data (encrypted documents, images, signatures, spreadsheets, etc.) with a business message.
- A SOAP message is XML, which means it can hold Unicode text, but not raw binary data.
  - Certain character codes are reserved or disallowed.
- It is possible to encode binary data as text in base64 format, but encoding and decoding these formats introduces overhead.
- In most cases, an XML document cannot even hold another XML document.
  - The prolog and DTD, if present, would have to be stripped off.
  - The character encoding of both documents would have to match.
- Soap With Attachments (SwA) is one solution for attaching arbitrary data to SOAP messages.
  - SwA was defined as an extension to the existing SOAP 1.1 standard.
  - SwA is a W3C note, submitted by Hewlett-Packard, IBM, and Microsoft.
- SwA appends documents to a SOAP message using the MIME format.
  - MIME has long been used to add attachments to email.
- There is a drawback to solutions of this kind; the attachments remain outside of the XML.
  - XML technologies (such as digital signatures) cannot be applied to the attachments.

 © 2011 ITCourseware, LLC

SwA generates a standard MIME compound object where the first part is the original SOAP message, and subsequent parts are attached documents.

The following example shows the general arrangement of parts in the MIME object. A unique separator string divides the data into parts. The first part is always the SOAP message. Subsequent parts each have a **Content-Id** identifier which can be used to reference them.

```
---------SEPARATOR_STRING------------
Content-Type: text/xml

<Envelope> ...SOAP MESSAGE... </Envel
ope>
---------SEPARATOR_STRING------------
Content-Type: image/jpeg
Content-Id: photo_attachment

.......ENCODED BINARY DATA ..........
.......ENCODED BINARY DATA ..........
.......ENCODED BINARY DATA ..........
.......ENCODED BINARY DATA ..........
---------SEPARATOR_STRING------------
```

Soap With Attachments (SwA) note: ***http://www.w3.org/TR/SOAP-attachments***

BUILDING ATTACHMENTS

- SAAJ supports SWA through the class **AttachmentPart**.
  - A **SOAPMessage** can hold both a **SOAPPart** (the actual SOAP message) and any number of **AttachmentPart**s.
- The wire format of the object changes from SOAP to MIME as soon as the first attachment is added.
- An **AttachmentPart** can be initialized with data from a string, a stream, a **Source** or a **DataHandler**.
- The programmer should also set the **Content-Type** and **Content-Id** headers.
- SAAJ allows specific attachments to be retrieved from messages by specifying their MIME headers.

 © 2011 ITCourseware, LLC

document-server/DocumentAttacher.java

```
...
public class DocumentAttacher extends GenericHandler {
   public boolean handleResponse(MessageContext context) {
      try {
         SOAPMessageContext soapContext = (SOAPMessageContext)context;
         SOAPMessage message = soapContext.getMessage();
         SOAPElement methodElement =
            (SOAPElement)message.getSOAPBody().getChildElements().next();
         SOAPElement resultElement =
            (SOAPElement)methodElement.getChildElements().next();
         String documentURL = resultElement.getValue();
         resultElement.removeContents();

         DataHandler dataHandler = new DataHandler(new URL(documentURL));
         AttachmentPart attachment =
            message.createAttachmentPart(dataHandler);
         attachment.setContentId("result-document");
         message.addAttachmentPart(attachment);
      }
      ...
}
```

Set the **Content-Id** header in the server.

document-server-client/DocServerClient.java

```
...
public class DocServerClient {
   public static void main(String[] args) throws Exception {
   ...
      MimeHeaders headers = new MimeHeaders();
      headers.addHeader("Content-Id", "result-document");
      Iterator it = response.getAttachments(headers);
      while (it.hasNext()) {
         AttachmentPart ap = (AttachmentPart)it.next();
         System.out.println("Contents of attachment: " );
         System.out.println(ap.getContent());
      }
   }
}
```

Find the attachment based upon the **Content-Id** header in the client.

A DOCUMENT SERVER

- To demonstrate the use of the JAX-RPC and SAAJ APIs, we will build a document server as a web service.
- The server must support downloading documents that are in non-XML format.
- JAX-RPC allows us to define the service endpoint interface and read the requests using Java syntax.
  - However, it is inconvenient to return the documents through JAX-RPC.
    - They would be sent as encoded strings with no type information.
- The documents will be sent as attachments on the reply messages.
- To add an attachment, we need access to the outgoing message in SAAJ format.
  - A message handler has this access.
- The application will not have a direct reference to the handler instance.
  - In fact, servers may pool multiple handler instances.
- To tell the handler which document to attach, the application will use the reply message itself.

 © 2011 ITCourseware, LLC

document-server/DocServerIF.java

```
...
public interface DocServerIF extends Remote {
   String getDocument(String documentName) throws RemoteException;
}
```

document-server/DocServerImpl.java

```
...
public class DocServerImpl implements DocServerIF {
   public String getDocument(String documentName) {
      return "file:///c:/PUB/" + documentName;
   }
}
```

Return a file-protocol URL string pointing to the file in the server's */PUB* directory.

document-server/DocumentAttacher.java

```
...
public class DocumentAttacher extends GenericHandler {
   public boolean handleResponse(MessageContext context) {
      try {
         SOAPMessageContext soapContext = (SOAPMessageContext)context;
         SOAPMessage message = soapContext.getMessage();
         SOAPElement methodElement =
            (SOAPElement)message.getSOAPBody().getChildElements().next();
         SOAPElement resultElement =
            (SOAPElement)methodElement.getChildElements().next();
         String documentURL = resultElement.getValue();
         resultElement.removeContents();

         DataHandler dataHandler = new DataHandler(new URL(documentURL));
         AttachmentPart attachment =
            message.createAttachmentPart(dataHandler);
         attachment.setContentId("result-document");
         message.addAttachmentPart(attachment);
      }
      catch(Exception e){
         e.printStackTrace();
      }
      return true;
   }
   ...
}
```

Drill down to the returned URL string.

Remove the string from the message, since it is of no interest to the client.

Use the URL to read the document, and add it as an attachment to the message.

Try It:

1. Run *buildanddeploy.bat* in the *document-server* directory.
2. Create a file called *thefile.txt* in the *c:\PUB* directory.
3. Compile and run *document-server-client/DocServerClient.java*.

LABS

In the *employee* directory you will find an employee lookup web service. The web service endpoint returns an **Employee** object when an integer **id** is passed into the **getEmployee()** method. An **Employee** has **id**, **firstName**, and **lastName** fields.

❶ Run *buildanddeploy.bat* in the *employee* directory. Invoke *build.bat* in the *employee/client* directory. Run *EmployeeClient.java*. Look at the request and response SOAP documents in the application server's log file.

❷ Find an image on the Internet for each of the employees. If you do not have internet access feel free to use the images provided in the *Solutions* directory.
(Solutions: *george_washington.gif, john_adams.gif, thomas_jefferson.gif, james_madison.gif, james_monroe.gif*)

❸ Enhance the employee lookup web service so that it returns back the employee's image as an attachment.
(Solution: *Attacher.java, config-interface.xml*)

❹ Enhance the current client so that it retrieves the attachment from the web service. You can save it to a file if you'd like, or just print it out. Hint: Use a client-side message handler.
(Solution: *client/Detacher.java, client/config.xml, client/build.bat*)

 © 2011 ITCourseware, LLC

Note:
The solution file *Attacher.java* expects the employee images to be located in the *c:/PUB* directory.