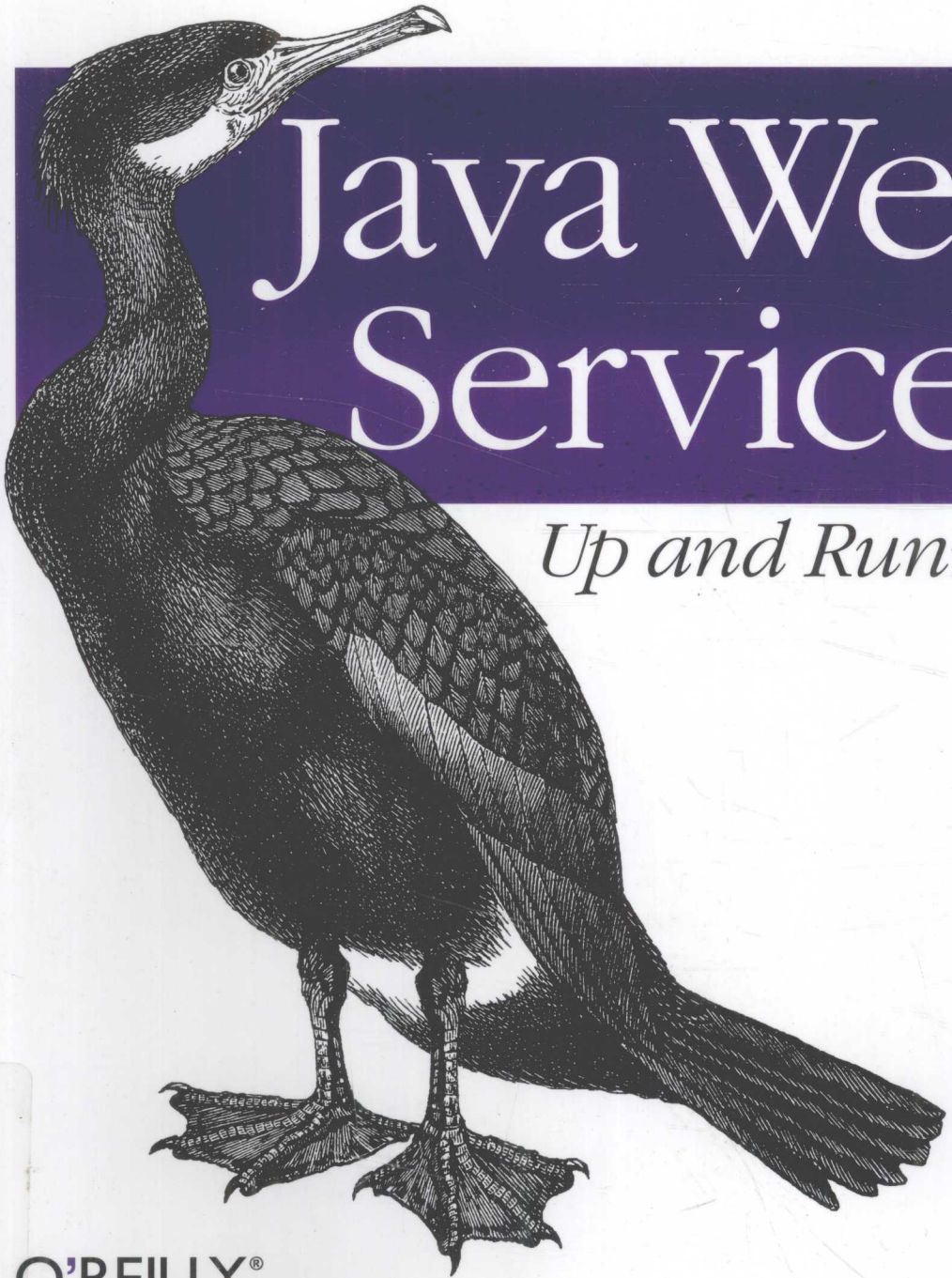
Java Web 服务：构建与运行（影印版）

Covers JAX-WS 2.1

Java Web Services

Up and Running



O'REILLY®
東南大學出版社

Martin Kalin 著

Java Web服务：构建与运行(影印版)

Java Web Services: Up and Running

Martin Kalin

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

Java Web服务：构建与运行：英文/（美）卡林（Kalin, M.）著. —影印本. —南京：东南大学出版社，2010.1

书名原文：Java Web Services: Up and Running

ISBN 978-7-5641-1927-0

Ⅰ. J… Ⅱ. 卡… Ⅲ. JAVA语言－程序设计－英文

Ⅳ. TP312

中国版本图书馆CIP数据核字（2009）第205659号

江苏省版权局著作权合同登记

图字：10-2009-243号

Java Web服务：构建与运行（影印版）

出版发行：东南大学出版社

地　　址：南京四牌楼2号　　邮编：210096

出 版 人：江　汉

网　　址：http://press.seu.edu.cn

电子邮件：press@seu.edu.cn

印　　刷：扬中市印刷有限公司

开　　本：787毫米×980毫米　16开本

印　　张：20印张

字　　数：336千字

版　　次：2010年1月第1版

印　　次：2010年1月第1次印刷

书　　号：ISBN 978-7-5641-1927-0

印　　数：1~1600册

定　　价：48.00元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

Preface

This is a book for programmers interested in developing Java web services and Java clients against web services, whatever the implementation language. The book is a code-driven introduction to *JAX-WS* (Java API for XML-Web Services), the framework of choice for Java web services, whether SOAP-based or REST-style. My approach is to interpret JAX-WS broadly and, therefore, to include leading-edge developments such as the Jersey project for REST-style web services, officially known as *JAX-RS* (Java API for XML-RESTful Web Services).

JAX-WS is bundled into the *Metro Web Services Stack*, or *Metro* for short. Metro is part of core Java, starting with Standard Edition 6 (hereafter, core Java 6). However, the Metro releases outpace the core Java releases. The current Metro release can be downloaded separately from *https://wsit.dev.java.net*. Metro is also integrated into the Sun application server, GlassFish. Given these options, this book's examples are deployed in four different ways:

Core Java only
: This is the low-fuss approach that makes it easy to get web services and their clients up and running. The only required software is the Java software development kit (SDK), core Java 6 or later. Web services can be deployed easily using the `Endpoint`, `HttpServer`, and `HttpsServer` classes. The early examples take this approach.

Core Java with the current Metro release
: This approach takes advantage of Metro features not yet available in the core Java bundle. In general, each Metro release makes it easier to write web services and clients. The current Metro release also indicates where JAX-WS is moving. The Metro release also can be used with core Java 5 if core Java 6 is not an option.

Standalone Tomcat
: This approach builds on the familiarity among Java programmers with standalone web containers such as Apache Tomcat, which is the reference implementation. Web services can be deployed using a web container in essentially the same way as are servlets, JavaServer Pages (JSP) scripts, and JavaServer Faces (JSF) scripts. A standalone web container such as Tomcat is also a good way to introduce container-managed security for web services.

GlassFish
: This approach allows deployed web services to interact naturally with other enterprise components such as Java Message Service topics and queues, a *JNDI* (Java Naming and Directory Interface) provider, a backend database system and the `@Entity` instances that mediate between an application and the database system, and an *EJB* (Enterprise Java Bean) container. The EJB container is important because a web service can be deployed as a stateless Session EJB, which brings advantages such as container-managed thread safety. GlassFish works seamlessly with Metro, including its advanced features, and with popular *IDE*s (Integrated Development Environment) such as NetBeans and Eclipse.

An appealing feature of JAX-WS is that the API can be separated cleanly from deployment options. One and the same web service can be deployed in different ways to suit different needs. Core Java alone is good for learning, development, and even lightweight deployment. A standalone web container such as Tomcat provides additional support. A Java application server such as GlassFish promotes easy integration of web services with other enterprise technologies.

Code-Driven Approach

My code examples are short enough to highlight key features of JAX-WS but also realistic enough to show off the production-level capabilities that come with the JAX-WS framework. Each code example is given in full, including all of the `import` statements. My approach is to begin with a relatively sparse example and then to add and modify features. The code samples vary in length from a few statements to several pages of source. The code is deliberately modular. Whenever there is a choice between conciseness and clarity in coding, I try to opt for clarity.

The examples come with instructions for compiling and deploying the web services and for testing the service against sample clients. This approach presents the choices that JAX-WS makes available to the programmer but also encourages a clear and thorough analysis of the JAX-WS libraries and utilities. My goal is to furnish code samples that can serve as templates for commercial applications.

JAX-WS is a rich API that is explored best in a mix of overview and examples. My aim is to explain key features about the architecture of web services but, above all, to illustrate each major feature with code examples that perform as advertised. Architecture without code is empty; code without architecture is blind. My approach is to integrate the two throughout the book.

Web services are a modern, lightweight approach to *distributed software systems*, that is, systems such as email or the World Wide Web that require different software components to execute on physically distinct devices. The devices can range from large servers through personal desktop machines to handhelds of various types. Distributed systems are complicated because they are made up of networked components. There

is nothing more frustrating than a distributed systems example that does not work as claimed because the debugging is tedious. My approach is thus to provide full, working examples together with short but precise instructions for getting the sample application up and running. All of the source code for examples is available from the book's companion site, at *http://www.oreilly.com/catalog/9780596521127*. My email address is *kalin@cdm.depaul.edu*. Please let me know if you find any code errors.

Chapter-by-Chapter Overview

The book has seven chapters, the last of which is quite short. Here is a preview of each chapter:

Chapter 1, *Java Web Services Quickstart*

This chapter begins with a working definition of *web services*, including the distinction between SOAP-based and REST-style services. This chapter then focuses on the basics of writing, deploying, and consuming SOAP-based services in core Java. There are web service clients written in Perl, Ruby, and Java to underscore the language neutrality of web services. This chapter also introduces Java's SOAP API and covers various ways to inspect web service traffic at the wire level. The chapter elaborates on the relationship between core Java and Metro.

Chapter 2, *All About WSDLs*

This chapter focuses on the service contract, which is a *WSDL* (Web Service Definition Language) document in SOAP-based services. This chapter covers the standard issues of web service style (`document` versus `rpc`) and encoding (`literal` versus `encoded`). This chapter also focuses on the popular but unofficial distinction between the *wrapped* and *unwrapped* variations of document style. All of these issues are clarified through examples, including Java clients against Amazon's E-Commerce services. This chapter explains how the *wsimport* utility can ease the task of writing Java clients against commercial web services and how the *wsgen* utility figures in the distinction between `document`-style and `rpc`-style web services. The basics of *JAX-B* (Java API for XML-Binding) are also covered. This chapter, like the others, is rich in code examples.

Chapter 3, *SOAP Handling*

This chapter introduces SOAP and logical handlers, which give the service-side and client-side programmer direct access to either the entire SOAP message or just its payload. The structure of a SOAP message and the distinction between SOAP 1.1 and SOAP 1.2 are covered. The messaging architecture of a SOAP-based service is discussed. Various code examples illustrate how SOAP messages can be processed in support of application logic. This chapter also explains how transport-level messages (for instance, the typical HTTP messages that carry SOAP payloads in SOAP-based web services) can be accessed and manipulated in JAX-WS. This chapter concludes with a section on JAX-WS support for

transporting binary data, with emphasis on *MTOM* (Message Transmission Optimization Mechanism).

Chapter 4, *RESTful Web Services*

This chapter opens with a technical analysis of what constitutes a REST-style service and moves quickly to code examples. The chapter surveys various approaches to delivering a Java-based RESTful service: `WebServiceProvider`, `HttpServlet`, Jersey Plain Old Java Object (POJO), and restlet among them. The use of a *WADL* (Web Application Definition Language) document as a service contract is explored through code examples. The *JAX-P* (Java API for XML-Processing) packages, which facilitate XML processing, are also covered. This chapter offers several examples of Java clients against real-world REST-style services, including services hosted by Yahoo!, Amazon, and Tumblr.

Chapter 5, *Web Services Security*

This chapter begins with an overview of security requirements for real-world web services, SOAP-based and REST-style. The overview covers central topics such as mutual challenge and message confidentiality, users-roles security, and WS-Security. Code examples clarify transport-level security, particularly under HTTPS. Container-managed security is introduced with examples deployed in the standalone Tomcat web container. The security material introduced in this chapter is expanded in the next chapter.

Chapter 6, *JAX-WS in Java Application Servers*

This chapter starts with a survey of what comes with a Java Application Server (JAS): an EJB container, a messaging system, a naming service, an integrated database system, and so on. This chapter has a variety of code examples: a SOAP-based service implemented as a stateless Session EJB, `WebService` and `WebServiceProvider` instances deployed through embedded Tomcat, a web service deployed together with a traditional website application, a web service integrated with *JMS* (Java Message Service), a web service that uses an `@Entity` to read and write from the Java DB database system included in GlassFish, and a WS-Security application under GlassFish.

Chapter 7, *Beyond the Flame Wars*

This is a very short chapter that looks at the controversy surrounding SOAP-based and REST-style web services. My aim is to endorse both approaches, either of which is superior to what came before. This chapter traces modern web services from DCE/RPC in the early 1990s through CORBA and DCOM up to the Java EE and .NET frameworks. This chapter explains why either approach to web services is better than the distributed-object architecture that once dominated in distributed software systems.

Freedom of Choice: The Tools/IDE Issue

Java programmers have a wide choice of productivity tools such as Ant and Maven for scripting and IDEs such as Eclipse, NetBeans, and IntelliJ IDEA. Scripting tools and IDEs increase productivity by hiding grimy details. In a production environment, such tools and IDEs are the sensible way to go. In a learning environment, however, the goal is to understand the grimy details so that this understanding can be brought to good use during the inevitable bouts of debugging and application maintenance. Accordingly, my book is neutral with respect to scripting tools and IDEs. Please feel free to use whichever tools and IDE suit your needs. My how-to segments go over code compilation, deployment, and execution at the command line so that details such as classpath inclusions and compilation/execution flags are clear. Nothing in any example depends on a particular scripting tool or IDE.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, filenames, file extensions, and emphasis.

`Constant width`
: Used for program listings as well as within paragraphs to refer to program elements such as variable or method names, data types, environment variables, statements, and keywords.

`Constant width bold`
: Used within program listings to highlight particularly interesting sections and in paragraphs to clarify acronyms.

This icon signifies a tip, suggestion, or general note.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Java Web Services: Up and Running*, by Martin Kalin. Copyright 2009 Martin Kalin, 978-0-596-52112-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596521127/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://www.oreilly.com/

Acknowledgments

Christian A. Kenyeres, Greg Ostravich, Igor Polevoy, and Ken Yu were kind enough to review this book and to offer insightful suggestions for its improvement. They made the book better than it otherwise would have been. I thank them heartily for the time and effort that they invested in this project. The remaining shortcomings are mine alone, of course.

I'd also like to thank Mike Loukides, my first contact at O'Reilly Media, for his role in shepherding my initial proposal through the process that led to its acceptance. Julie Steele, my editor, has provided invaluable support and the book would not be without her help. My thanks go as well to the many behind-the-scenes people at O'Reilly Media who worked on this project.

This book is dedicated to Janet.

Table of Contents

CHAPTER 1

Java Web Services Quickstart

What Are Web Services?

Although the term *web service* has various, imprecise, and evolving meanings, a glance at some features typical of web services will be enough to get us into coding a web service and a client, also known as a consumer or requester. As the name suggests, a web service is a kind of *webified application*, that is, an application typically delivered over *HTTP* (Hyper Text Transport Protocol). A web service is thus a distributed application whose components can be deployed and executed on distinct devices. For instance, a stock-picking web service might consist of several code components, each hosted on a separate business-grade server, and the web service might be consumed on PCs, handhelds, and other devices.

Web services can be divided roughly into two groups, *SOAP*-based and *REST*-style. The distinction is not sharp because, as a code example later illustrates, a SOAP-based service delivered over HTTP is a special case of a REST-style service. *SOAP* originally stood for Simple Object Access Protocol but, by serendipity, now may stand for Service Oriented Architecture (SOA) Protocol. Deconstructing SOA is nontrivial but one point is indisputable: whatever SOA may be, web services play a central role in the SOA approach to software design and development. (This is written with tongue only partly in cheek. SOAP is officially no longer an acronym, and SOAP and SOA can live apart from one another.) For now, SOAP is just an *XML* (EXtensible Markup Language) dialect in which documents are messages. In SOAP-based web services, the SOAP is mostly unseen infrastructure. For example, in a typical scenario, called the request/response *message exchange pattern* (MEP), the client's underlying SOAP library sends a SOAP message as a service request, and the web service's underlying SOAP library sends another SOAP message as the corresponding service response. The client and the web service source code may provide few hints, if any, about the underlying SOAP (see Figure 1-1).

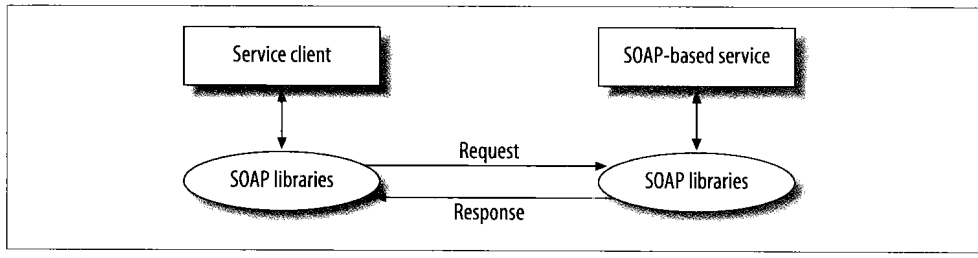


Figure 1-1. Architecture of a typical SOAP-based web service

REST stands for REpresentational State Transfer. Roy Fielding, one of the main authors of the HTTP specification, coined the acronym in his Ph.D. dissertation to describe an architectural style in the design of web services. SOAP has standards (under the World Wide Web Consortium [W3C]), toolkits, and bountiful software libraries. REST has no standards, few toolkits, and meager software libraries. The REST style is often seen as an antidote to the creeping complexity of SOAP-based web services. This book covers SOAP-based and REST-style web services, starting with the SOAP-based ones.

Except in test mode, the client of either a SOAP-based or REST-style service is rarely a web browser but rather an application without a graphical user interface. The client may be written in any language with the appropriate support libraries. Indeed, a major appeal of web services is language transparency: the service and its clients need not be written in the same language. Language transparency is the key to web service *interoperability*; that is, the ability of web services and requesters to interact seamlessly despite differences in programming languages, support libraries, and platforms. To underscore this appeal, clients against our Java web services will be written in various languages such as C#, Perl, and Ruby, and Java clients will consume services written in other languages, including languages unknown.

There is no magic in language transparency, of course. If a SOAP-based web service written in Java can have a Perl or a Ruby consumer, there must be an intermediary that handles the differences in data types between the service and the requester languages. XML technologies, which support structured document interchange and processing, act as the intermediary. For example, in a typical SOAP-based web service, a client transparently sends a SOAP document as a request to a web service, which transparently returns another SOAP document as a response. In a REST-style service, a client might send a standard HTTP request to a web service and receive an appropriate XML document as a response.

Several features distinguish web services from other distributed software systems. Here are three:

Open infrastructure
: Web services are deployed using industry-standard, vendor-independent protocols such as HTTP and XML, which are ubiquitous and well understood. Web services

can piggyback on networking, data formatting, security, and other infrastructures already in place, which lowers entry costs and promotes interoperability among services.

Language transparency
: Web services and their clients can interoperate even if written in different programming languages. Languages such as C/C++, C#, Java, Perl, Python, Ruby, and others provide libraries, utilities, and even frameworks in support of web services.

Modular design
: Web services are meant to be modular in design so that new services can be generated through the integration and layering of existing services. Imagine, for example, an inventory-tracking service integrated with an online ordering service to yield a service that automatically orders the appropriate products in response to inventory levels.

What Good Are Web Services?

This obvious question has no simple, single answer. Nonetheless, the chief benefits and promises of web services are clear. Modern software systems are written in a variety of languages—a variety that seems likely to increase. These software systems will continue to be hosted on a variety of platforms. Institutions large and small have significant investment in legacy software systems whose functionality is useful and perhaps mission critical; and few of these institutions have the will and the resources, human or financial, to rewrite their legacy systems.

It is rare that a software system gets to run in splendid isolation. The typical software system must interoperate with others, which may reside on different hosts and be written in different languages. Interoperability is not just a long-term challenge but also a current requirement of production software.

Web services address these issues directly because such services are, first and foremost, language- and platform-neutral. If a legacy COBOL system is exposed through a web service, the system is thereby interoperable with service clients written in other programming languages.

Web services are inherently *distributed* systems that communicate mostly over HTTP but can communicate over other popular transports as well. The communication payloads of web services are structured text (that is, XML documents), which can be inspected, transformed, persisted, and otherwise processed with widely and even freely available tools. When efficiency demands it, however, web services also can deliver binary payloads. Finally, web services are a work in progress with real-world distributed systems as their test bed. For all of these reasons, web services are an essential tool in any modern programmer's toolbox.