Part III Web Services

Part III explores web services.

This part contains the following chapters:

- Chapter 17, Introduction to Web Services
- Chapter 18, Building Web Services with JAX-WS
- Chapter 19, Building RESTful Web Services with JAX-RS
- Chapter 20, Advanced JAX-RS Features
- Chapter 21, Running the Advanced JAX-RS Example Application

Introduction to Web Services

Part III of the tutorial discusses Java EE 6 web services technologies.

For this book, these technologies include Java API for XML Web Services (JAX-WS) and Java API for RESTful Web Services (JAX-RS).

The following topics are addressed here:

- What Are Web Services?
- Types of Web Services
- Deciding Which Type of Web Service to Use

What Are Web Services?

Web services are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP).

As described by the World Wide Web Consortium (W3C), web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks.

Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions, thanks to the use of XML.

Web services can be combined in a loosely coupled way to achieve complex operations.

Programs providing simple services can interact with each other to deliver sophisticated added-value services.

Types of Web Services

On the conceptual level, a service is a software component provided through a network-accessible endpoint.

The service consumer and provider use messages to exchange invocation request and response information in the form of self-containing documents that make very few assumptions about the technological capabilities of the receiver.

On a technical level, web services can be implemented in various ways.

The two types of web services discussed in this section can be distinguished as “big” web services and “RESTful” web services.

“Big” Web Services

In Java EE 6, JAX-WS provides the functionality for “big” web services, which are described in Chapter 18, Building Web Services with JAX-WS.

Big web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats.

Such systems often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

The SOAP message format and the WSDL interface definition language have gained widespread adoption.

Many development tools, such as NetBeans IDE, can reduce the complexity of developing web service applications.

A SOAP-based design must include the following elements.

- A formal contract must be established to describe the interface that the web service offers.

WSDL can be used to describe the details of the contract, which may include messages, operations, bindings, and the location of the web service.

You may also process SOAP messages in a JAX-WS service without publishing a WSDL.

- The architecture must address complex nonfunctional requirements.

Many web service specifications address such requirements and establish a common vocabulary for them.

Examples include transactions, security, addressing, trust, coordination, and so on.

- The architecture needs to handle asynchronous processing and invocation.

In such cases, the infrastructure provided by standards, such as Web Services Reliable Messaging (WSRM), and APIs, such as JAX-WS, with their client-side asynchronous invocation support, can be leveraged out of the box.

RESTful Web Services

In Java EE 6, JAX-RS provides the functionality for Representational State Transfer (RESTful) web services.

REST is well suited for basic, ad hoc integration scenarios.

RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service–API definitions.

Project Jersey is the production-ready reference implementation for the JAX-RS specification.

Jersey implements support for the annotations defined in the JAX-RS specification, making it easy for developers to build RESTful web services with Java and the Java Virtual Machine (JVM).

Because RESTful web services use existing well-known W3C and Internet Engineering Task Force (IETF) standards (HTTP, XML, URI, MIME) and have a lightweight infrastructure that allows services to be built with minimal tooling, developing RESTful web services is inexpensive and thus has a very low barrier for adoption.

You can use a development tool such as NetBeans IDE to further reduce the complexity of developing RESTful web services.

A RESTful design may be appropriate when the following conditions are met.

- The web services are completely stateless.

A good test is to consider whether the interaction can survive a restart of the server.

- A caching infrastructure can be leveraged for performance.

If the data that the web service returns is not dynamically generated and can be cached, the caching infrastructure that web servers and other intermediaries inherently provide can be leveraged to improve performance.

However, the developer must take care because such caches are limited to the HTTP `GET` method for most servers.

- The service producer and service consumer have a mutual understanding of the context and content being passed along.

Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it meaningfully.

In the real world, most commercial applications that expose services as RESTful implementations also distribute so-called value-added toolkits that describe the interfaces to developers in popular programming languages.

- Bandwidth is particularly important and needs to be limited.

REST is particularly useful for limited-profile devices, such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.

- Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style.

Developers can use such technologies as JAX-RS and Asynchronous JavaScript with XML (AJAX) and such toolkits as Direct Web Remoting (DWR) to consume the services in their web applications.

Rather than starting from scratch, services can be exposed with XML and consumed by HTML pages without significantly refactoring the existing web site architecture.

Existing developers will be more productive because they are adding to something they are already familiar with rather than having to start from scratch with new technology.

RESTful web services are discussed in Chapter 19, Building RESTful Web Services with JAX-RS.

This chapter contains information about generating the skeleton of a RESTful web service using both NetBeans IDE and the Maven project management tool.

Deciding Which Type of Web Service to Use

Basically, you would want to use RESTful web services for integration over the web and use big web services in enterprise application integration scenarios that have advanced quality of service (QoS) requirements.

- **JAX-WS: addresses advanced QoS requirements commonly occurring in enterprise computing.**

When compared to JAX-RS, JAX-WS makes it easier to support the WS-* set of protocols, which provide standards for security and reliability, among other things, and interoperate with other WS-* conforming clients and servers.

- JAX-RS: makes it easier to write web applications that apply some or all of the constraints of the REST style to induce desirable properties in the application, such as loose coupling (evolving the server is easier without breaking existing clients), scalability (start small and grow), and architectural simplicity (use off-the-shelf components, such as proxies or HTTP routers).

You would choose to use JAX-RS for your web application because it is easier for many types of clients to consume RESTful web services while enabling the server side to evolve and scale.

Clients can choose to consume some or all aspects of the service and mash it up with other web-based services.

Note - For an article that provides more in-depth analysis of this issue, see "RESTful Web Services vs.

"Big" Web Services: Making the Right Architectural Decision," by Cesare Pautasso, Olaf Zimmermann, and Frank Leymann from *WWW '08: Proceedings of the 17th International Conference on the World Wide Web* (2008), pp. 805–814.