

6. IMPLEMENTING SOAP SERVICES BY USING JAX-WS

6.1. Implementing SOAP Services by Using JAX-WS

Objectives

After completing this lesson, you should be able to handle communications with SOAP Services, including how to:

- Describe a SOAP Web Service structure
- Create SOAP Web Services using JAX-WS API
- Create SOAP Web Service clients




 Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Echemos un vistazo a las formas de implementar servicios web SOAP con el uso de la API JAX-WS. En este capítulo, estamos viendo una estructura de servicio SOAP. También estamos analizando las formas en que podríamos respaldar la asignación de los artefactos del servicio SOAP al código Java utilizando la API JAX-WS. Y, por supuesto, también las formas de escribir clientes que invoquen dichos servicios.

WebServices and SOAP


SOAP WebServices


- **Web Services** are interoperable, platform independent mechanism for component interactions.
- SOAP decouples message representation from transport.
- **Service Interface** is described with WSDL and XSD files.
- **Service Implementation** are disguised — **Consumer** is unaware of implementation details of the service.



SOAP Message

- Can be transported over different protocols
- Must be enclosed into **SOAP Envelope**
- May have **Headers**, such as security, reliability, and so on
- Must have a **Body** with business data, or **Fault** elements
- May have attachments



 Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

There are two implementation styles of SOAP communications - RPC and Document.

- An RPC style service produces SOAP messages that contain the name of the service operation and its parameters.
- A Document style service simply relies on XML Schema to describe transmitted SOAP message body.

Document style is a recommended approach and RPC style is supported for backward compatibility reasons.

All examples in this course use Document style services.

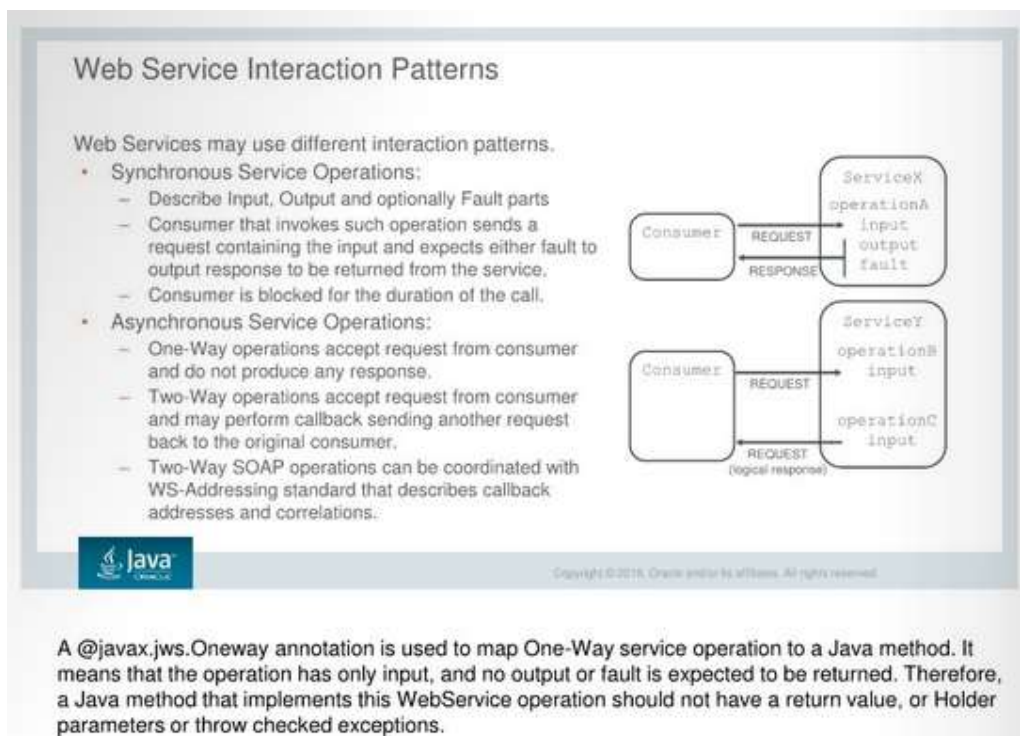
Comencemos con los servicios web y su uso del protocolo SOAP. El propósito de los servicios web SOAP es crear un mecanismo independiente de plataforma interoperable para intercambiar llamadas de información entre sistemas. Es posible que los sistemas en sí mismos no estén escritos en Java. Pueden estar escritos en cualquier lenguaje de programación. Ese no es el punto. Por supuesto, podemos escribirlos en Java.

El objetivo de los servicios es estandarizar dichos intercambios independientemente del lenguaje de programación que se utilice para implementar un servicio o el cliente. Para lograrlo, el servicio SOAP debe basarse en una estandarización estricta de una interfaz de servicio y un protocolo de transporte que se utiliza para comunicar mensajes. El transporte subyacente podría ser cualquier cosa, en realidad. Por lo general, podría ser HTTP, pero no tiene por qué serlo. Puede haber otros protocolos de transporte.

Pero la estructura del mensaje que envía a ese transporte está estrictamente estandarizada por el estándar de servicios SOAP. Entonces, el mensaje contendrá un sobre SOAP, que es un contenedor XML, un elemento que envuelve el resto del código del mensaje. Puede contener una serie de encabezados, cuerpo del mensaje.

El cuerpo del mensaje puede contener cualquier contenido que desee enviar, o quizás fallas si desea informar algún tipo de error. Y opcionalmente, el mensaje podría venir con archivos adjuntos. Entonces, si desea adjuntar documentos adicionales. Bueno, sobre todo supongo que estarás transportando el contenido del mensaje dentro del cuerpo.

Ahora, el programa que envía mensajes, llamando a dichos servicios, se conoce como consumidor de servicios. El programa que procesa los mensajes, devuelve algunos otros mensajes y maneja las llamadas de servicio se llama implementación del servicio. Entonces, el consumidor a través de cualquier protocolo de transferencia produce administradores SOAP. El productor del servicio que tenga una implementación del servicio entre bastidores procesará dichos mensajes.



Ahora, puede haber diferentes patrones de interacción asociados con los servicios SOAP. En la práctica, diría dos tipos principales, síncrono versus asíncrono, aunque también hay algunos tipos dentro de las entregas de mensajes asíncronos.

La entrega síncrona significa que envía una solicitud a un servicio y espera que el servicio le devuelva algún resultado o error. Así que esa es la interacción. En este caso, el consumidor será bloqueado mientras dure la llamada. El consumidor enviará una solicitud y esperará a que se devuelva la salida o la falla. La alternativa es la interacción asíncrona. En ese caso, un servicio define una operación que tiene entrada, pero no tiene salida ni elementos de falla.

Así que podríamos ser de una sola manera, dispara y olvida el tipo de interacción en la que tu consumidor envía la solicitud al servicio SOAP y eso es todo. Simplemente no esperaba nada a cambio. Multa.

Sin embargo, existe la posibilidad de implementar una interacción SOAP bidireccional con estilo asíncrono donde la segunda operación se utiliza para realizar una devolución de llamada. Entonces, en cierto sentido, el consumidor en este caso también es un servicio porque puede recibir la llamada, no solo enviar la llamada. Y la idea de una devolución de llamada es que es una solicitud nueva. Técnicamente hablando desde el punto de vista del protocolo, no es realmente una respuesta. Es solo otra petición.

Pero esa solicitud puede tratarse lógicamente como si fuera una respuesta a la solicitud recibida anteriormente. La idea es que podrías poner una identidad de la solicitud en esta primera llamada aquí mismo. Cuando estás enviando la solicitud. Puede asociar esta solicitud con un encabezado denominado encabezado de direccionamiento WS. Específicamente la parte que se ocupa de la identificación de correlación.

Luego, también coloca otro encabezado, que es una dirección de devolución de llamada. La dirección de devolución de llamada identifica a dónde enviar esa llamada, la devolución de llamada, a dónde debe ir. Y la identificación de correlación simplemente se puede copiar desde la solicitud allí a la solicitud aquí.

Entonces, este consumidor, cuando recibe esa solicitud, la examina y dice: ¿cuál es el valor de ID de correlación? Y lo compara con el ID de correlación que utilizó para enviar la solicitud inicial. Y así es como los une. Son por lo demás dos llamadas completamente independientes. Pero pueden vincularse en un par con la identidad de ID de correlación para la solicitud.

Web Service Interface

Web Services Description Language (WSDL) is an XML-based standard for describing SOAP Web Services interfaces.

- XML Schema Definition (XSD) documents describe data structures that services can use.
- WSDL file describes the service with:
 - **Abstract WSDL part:**
 - References XSD file to use data-structures it defines
 - Describes messages comprising elements from the XSD that this service would accept and return in its operations
 - Describes port types - logical groups of operations
 - Describes operations that would use messages to produce input, output, or fault parts
 - **Concrete WSDL part:**
 - Describes bindings to define how to invoke operations
 - Describes service to define an end-point invocation address where this service can be found

ProductQuote.xsd

```
<element>...</element>
<element>...</element>
```

ProductQuoteService.wsdl

```
<types>import xsd</types>
<message>...</message>
<message>...</message>
<portType>
  <operation>...</operation>
  <operation>...</operation>
</portType>
<binding>...</binding>
<service>...</service>
```

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Los servicios SOAP deben describirse con los documentos de lenguaje de descripción de servicios web, archivos WSDL. Son descriptores base XML de servicios. También usan archivos de definición de esquemas XML, archivos XSD, para describir tipos de datos y estructuras de datos que los servicios usan para formular mensajes, enviar y recibir contenido.

Un archivo WSDL, lo primero que hace es importar el esquema previamente definido. Entonces dice que voy a usar esa colección de tipos XML, elementos, descritos en un esquema. Luego, el archivo WSDL utilizará los elementos del esquema en los mensajes que define. Los mensajes van a ser utilizados por las operaciones, y luego estas operaciones estarán disponibles para que usted las invoque.

Un archivo WSDL está estructurado en dos partes llamadas WSDL abstracto y concreto. No son archivos diferentes. Sigue siendo el mismo archivo, pero lógicamente son dos partes de ese archivo. Un WSDL abstracto describe cuáles son los mensajes, cuáles son las operaciones, cómo se agrupan las operaciones. Un grupo de operaciones se denomina tipo de puerto y es un grupo lógico de operaciones. Entonces, cómo se agrupan.

Un WSDL concreto describe la forma en que implementa ese servicio con respecto a un protocolo de transporte específico, qué transporte va a utilizar y dónde va a implementar ese servicio. Entonces, la parte abstracta describe las capacidades generales del servicio. La parte concreta describe el despliegue físico de ese servicio.

XML Schema Definition

XML Schema Definition document describes data-structures that web services can use.

- Define with a **unique namespace**.
- It describes a number of elements, types, and so on.
- Components described by the schema can be mapped to Java Classes using **Java Architecture for XML Binding (JAXB)** API with annotations or XML descriptors.

```
<?xml version="1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/demos/ProductQuote"
elementFormDefault="qualified">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:int"/>
        <xs:element name="quantity" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ...
</xs:schema>
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "id",
    "quantity"
})
@XmlRootElement(name = "Product")
public class Product {
    protected int id;
    protected int quantity;
    ...
}
```

✚ JAXB API is covered in the Appendix E



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Primero, comencemos con el archivo de definición del esquema. Las definiciones de esquema XML describen las estructuras de datos que le gustaría usar en su aplicación, específicamente en este caso para los servicios SOAP. Este es el esquema que describe algún tipo de producto con ID y cantidad. Para mapear este documento de esquema XML a código Java, usamos la API llamada JAXB, Java Architecture for XML Binding, que se cubre en el Apéndice E. Más información sobre eso en el Apéndice E, cómo funcionan exactamente estos mapeos.

Sin embargo, el ejemplo básico está directamente presente en este PowerPoint. Como puede ver, estamos mapeando aquí el producto del elemento raíz XML. Ahí tienes Ahí vas. Asignamos ese producto de clase Java a ese esquema. Pero hay más, por eso tenemos un apéndice separado. Así que vamos a echar un vistazo a otros casos de estas asignaciones.

Pero supongamos que aquí es donde comienzas. Cuando esté pensando en qué tipo de datos le gustaría que procesen sus servicios web, debe describir los datos en XML y asignarlos a su código Java. Una vez más, ¿por qué necesita describir los datos en XML?

Porque su invocador de un servicio no está necesariamente escrito en Java. Podría usar cualquier otro lenguaje de programación, entonces, ¿qué sabe sobre las clases de Java? Nada. La única forma en que puede decirles lo que espera que envíen, lo que espera que manejen en los valores de retorno, es describiendo sus estructuras de datos en XML.

WSDL Schemas and Namespaces

Web Services Description Language (WSDL) namespace definitions

- References XSD via its **unique namespace** to access data structures described by the XML Schema
- Defines its own **unique namespace**, for referencing components described within this WSDL file
- Each namespace can be assigned an arbitrary prefix **tns** and so on

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PriceQuote" targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService"
  xmlns:tns="http://xmlns.oracle.com/demos/PriceQuoteService"
  xmlns:ns1="http://xmlns.oracle.com/demos/ProductQuote"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://xmlns.oracle.com/demos/ProductQuote"
        schemaLocation="ProductQuote.xsd"/>
    </xsd:schema>
  </types>
  ...
</definitions>
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Ahora, el siguiente concepto importante con el que debe sentirse cómodo es el concepto de un espacio de nombres XML. Verá, tiene que controlar la unicidad de los nombres de elementos XML, y la unicidad de un nombre de elemento XML se controla con la ayuda del concepto de espacio de nombres. Cuando describe un WSDL, un esquema, cualquier documento XML, tiene la capacidad de describir un espacio de nombres de destino para este documento. Ahí está.

¿Podemos retroceder una página? Espacio de nombres de destino para un esquema. ¿Darse cuenta de? Ahí tienes

Este espacio de nombres parece una URL. Pero créeme, no lo es. Es una etiqueta de texto arbitraria. ¿Por qué parece una URL? La única razón por la que se vería como una URL es porque esta es la forma de garantizar su unicidad universal. Un nombre de dominio de Internet es único en primer lugar.

Entonces, por ejemplo, si trabajo para Oracle, elijo usar Oracle.com como mi dominio. Y luego, presumiblemente dentro de la empresa, debería poder acordar algún tipo de convención de nombres. Así que demostraciones de código de producto. Cualquiera que sea la convención de nomenclatura adoptada dentro de su empresa.

Línea de fondo. Para cada tipo de documento XML que esté utilizando, como esquema, WSDL, lo que sea, sea cual sea el tipo de documento, debe crear un espacio de nombres único que identifique ese documento. Así que ese es un identificador único de ese esquema. Y ese es un identificador único de algún archivo WSDL.

Observe que estoy importando el archivo de esquema aquí. Así que estoy importando la cotización del producto XSD. Y digo que esta cita de producto tiene este espacio de nombres. Si recuerda, esa era la URL, bueno, URL, etiqueta de texto, para el espacio de nombres que usamos en ese esquema.

Luego puede asociar el espacio de nombres con un prefijo. El prefijo es absolutamente arbitrario. No tienes que llamarlo de ninguna manera en particular. Llámelo NS1, lo que sea, como en el espacio de nombres uno, dos, tres. Lo que sea. El nombre no es significativo.

Entonces, cualquier elemento en este documento que tenga un prefijo en NS1 provendrá de ese esquema. Así te refieres a ellos. O no sé, cualquier cosa descrita en este documento con ese espacio de nombres tendrá como prefijo las palabras TNS.

Aquí hay otro ejemplo para ti. Mira, este es un lenguaje de esquema. Este es un lenguaje de esquema. El idioma con el que describe los esquemas XML y ve el prefijo es XSD. Bueno, allá vas. Eso significa que estás hablando de elemento, importación o esquema de elemento que proviene de ese lenguaje de esquema.

¿Por qué es eso importante? Podría haber definido un elemento llamado esquema en su esquema. ¿Cómo diferencia su esquema de elementos del esquema de elementos de un lenguaje de esquema? Por su prefijo. Y el prefijo está asociado con un espacio de nombres único.

Así que todos estos prefijos diferentes, SOAP, lo que sea, todos provienen de espacios de nombres relevantes. Bueno, ahora tenemos algunos elementos que no tienen prefijo. Elemento de definiciones. Elemento de tipos. Bueno, ¿por qué no tienen prefijo? Es porque el espacio de nombres del WSDL está asociado con XMLNS.

No XMLNS llamando a algo. Solo XMLNS. Y ese es un espacio de nombres predeterminado. Eso significa que cualquier elemento que no tenga un prefijo explícito al frente pertenece a ese espacio de nombres. Ahí tienes Entonces no son URL reales. Son solo identificadores de espacios de nombres.

WSDL Messages, PortTypes, and Operations

Abstract WSDL describes capabilities of the service:

- Messages, which comprise parts referencing elements described by XSDs
- Port Types, which define logical groups of operations
- Operations, which define:
 - input
 - output
 - faultreferencing previously defined messages

```
<?xml version="1.0"?>
<definitions ...>
  <types .../>
  <message name="QuoteRequest">
    <part name="request" element="ns1:Product"/>
  </message>
  <message name="QuoteResponse">
    <part name="response" element="ns1:Quote"/>
  </message>
  <message name="QuoteFault">
    <part name="response" element="ns1:QuoteError"/>
  </message>
  <portType name="PriceQuote">
    <operation name="getQuote">
      <input message="tns:QuoteRequest"/>
      <output message="tns:QuoteResponse"/>
      <fault name="QuoteFault" message="tns:QuoteFault"/>
    </operation>
  </portType>
</definitions>
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Echemos un vistazo más a fondo. Así que hemos importado algunos esquemas. Eso se hizo en el elemento de tipos. Y seguimos con el archivo WSDL. Esta vez estamos viendo mensajes. Un mensaje utiliza contenido del esquema: NS1, recuerde que es un prefijo de esquema.

Así que imagine que en mi archivo de esquema describí un elemento llamado producto. Describí un elemento llamado cotización. Describí un elemento llamado error de cotización. Y ahora me gustaría usarlos como mensajes que enviaré y recibiré como parte de mi entrada, salida o falla de cualquier operación de servicio web. El mismo esquema XML puede ser usado por diferentes servicios, y es por eso que un servicio puede describir la forma en que se usan los elementos del esquema colocándolos en el contexto de los mensajes.

Los mensajes son entradas, salidas y fallas de cualquier operación. Operaciones agrupadas en tipos de puerto. Ahí tienes Hay un grupo de operaciones. Y eso concluye la parte abstracta del WSDL.

WSDL Bindings and Services

Concrete WSDL describes access to the service:

- Bindings, which map port types and operations they define to transport protocol
- Service element, which describes locations where ports were deployed

```
<definitions ...>
  ...
  <binding name="PriceQuoteSOAP" type="tns:PriceQuote">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getQuote">
      <input>
        <soap:body use="literal" parts="request"/>
      </input>
      <output>
        <soap:body use="literal" parts="response"/>
      </output>
      <fault name="QuoteFault">
        <soap:fault use="literal" name="QuoteFault"/>
      </fault>
    </operation>
  </binding>
  <service name="PriceQuote">
    <port name="PriceQuote" binding="tns:PriceQuoteSOAP">
      <soap:address location="http://localhost:7001/services/PriceQuote"/>
    </port>
  </service>
</definitions>
```



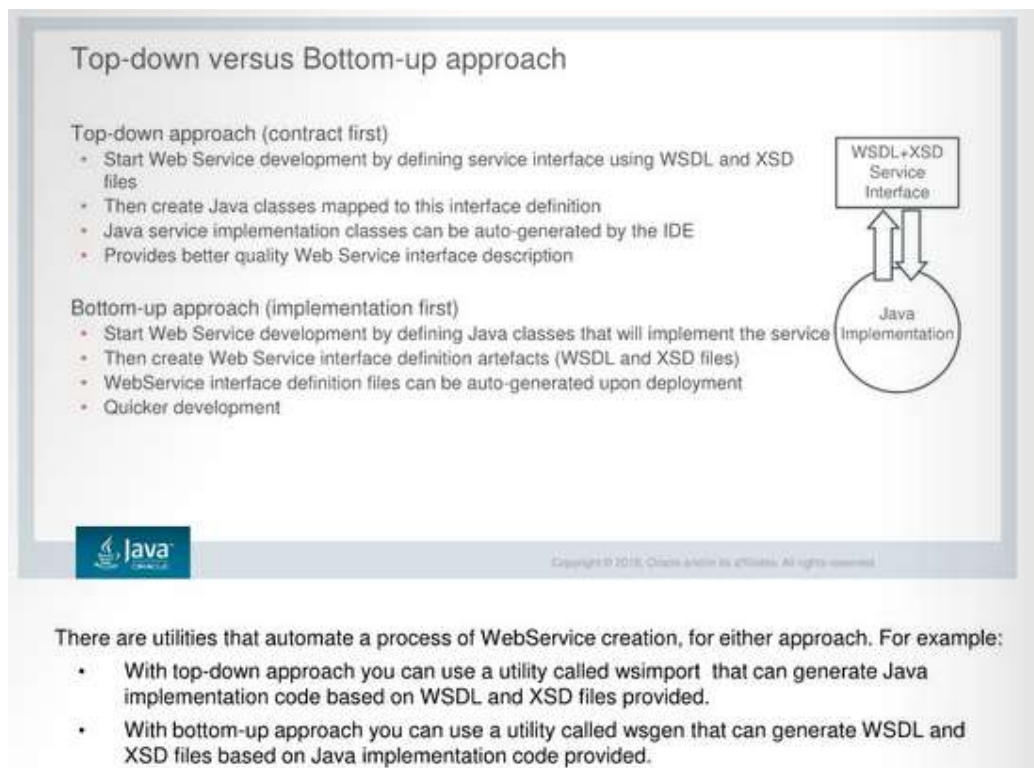
Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Procedemos a la parte WSDL concreta. Estamos hablando de implementar ese tipo de puerto sobre un protocolo de transporte específico. Entonces, ese tipo de puerto se implementó sobre SOAP HTTP.

Estamos diciendo exactamente cómo nos gustaría que se manejaran las asignaciones al protocolo SOAP HTTP. En este caso particular, estamos diciendo estilo Documento. Hay dos estilos, llamados Documento y hay otro estilo llamado RPC. Un estilo ligeramente diferente de envolver parámetros. RPC es un estilo heredado. Se supone que debe usarlo para compatibilidad con versiones anteriores de servicios de estilo anterior. Actualmente, se

recomienda el estilo Documento, que básicamente transmite el mensaje tal como está en el cuerpo del paquete SOAP.

Así transmitiendo mensajes. Ah, y esta parte define el lugar donde se implementa el servicio. Ubicación, en qué URL se implementa el servicio. Así que espero que este conjunto de páginas te haya ayudado a sortear los WSDL y los archivos de esquema.



Ahora, ¿cómo se diseña un servicio Java SOAP? Puede comenzar con WSDL y esquemas, definir la interfaz y luego crear clases Java que se asignan a estos WSDL y esquemas. O puedes hacerlo al revés. Comience creando clases Java, coloque algunas anotaciones en ellas y produzca WSDL y esquemas. Entonces, el primer enfoque se llama de arriba hacia abajo. Este último se llama de abajo hacia arriba.

Personalmente, normalmente recomendaría usar de arriba hacia abajo. El enfoque de abajo hacia arriba, el primer enfoque de implementación, podría ser más rápido de desarrollar porque simplemente dice que esto se clasifica como un servicio web. Eso es todo. Generarme un WSDL. Y te generará un WSDL.

Pero la calidad de ese WSDL y esquema es una incógnita. Y, en última instancia, ese WSDL y esquema es lo que expondrá su servicio al resto del mundo. Probablemente desee tener una interfaz de arriba hacia abajo limpia y bien diseñada donde primero pensó en cómo el servicio web será visible para el resto del mundo, y luego descubra cómo asignarlo a su código Java en lugar de su WSDL. y los esquemas serán producto de un accidente que de un diseño en un enfoque de abajo hacia arriba. Pero depende Quiero decir, de abajo hacia arriba es sin duda más rápido de desarrollar.

Map Java Interface to WSDL

Map WSDL operations to java Interface using Java for XML Web Services (JAX-WS) API.

- Define **Java Interface** mapped to the **Web Service** described by WSDL file.
- Define **SOAP protocol binding**.
- Map **Java operations** to **WSDL operations** with the **WebMethod** annotation.
- Describe mappings for operation **return type** and **parameters**.
- Declare **exceptions** mapped to fault elements of the service operation.

```
...
@WebService(name="PriceQuote", targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService")
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
@XmlSeeAlso({ObjectFactory.class})
public interface PriceQuote {
    @WebMethod
    @WebResult(name="Quote",
        targetNamespace="http://xmlns.oracle.com/demos/ProductQuote",
        partName="response")
    public List getQuote(@WebParam(name = "Product",
        targetNamespace = "http://xmlns.oracle.com/demos/ProductQuote",
        partName = "request") Product request) throws QuoteFault;
}
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

JAX-WS Requirements for Web Service Methods

- Must be public. By default, every public method in the class will be a part of the web service.
- Must not be static or final
- Must have JAXB-compatible parameters and return types. Parameters and return types must not implement the java.rmi.Remote interface.

WebMethod annotations used to map Java method to WSDL operation. It has following properties:

- **operationName:** Name of the wsdl:operation matching this method
- **exclude:** Marks a method to NOT be exposed as a web method
- **action:** Associate this operation with the SOAP Action property. Unique label for this operation, that may optionally be passed as a SOAP Header to link client call to specific operation.

The SOAPBinding annotation defines parameter style mapping for the web service. It could have two values:

- **BARE:** Method parameters represent the entire message body.
- **WRAPPED:** Parameters are elements wrapped inside a top-level element named after the operation.

The WebResult annotation defines the datatype mapping between the return type of the Java operation and output message used by the operation described within the wsdl file.

The WebParam annotation defines mapping between parameters of the Java Method and input message used by the operation described within the wsdl file.

XMLSeeAlso annotation instructs JAXB API to bind other classes (in this case ObjectFactory.class) when binding this (PriceQuote.class).

ObjectFactory class contains helper methods to produce java objects (Product, Quote, QuoteFault) mapped to input, output, and fault elements used by operation getQuote that this interface maps for the service implementation.

Para mapear el código Java al archivo WSDL, puede comenzar mapeando una interfaz. Y en la interfaz aquí, veo la anotación y el servicio web. Y espero que ahora te hagas una idea de lo que está pasando, porque estamos diciendo qué espacio de nombres se va a utilizar. Eso es básicamente asignarlo a un espacio de nombres WSDL específico.

Describir qué enlaces SOAP usaremos. Es un estilo simple de documento. Esta "ver también fábrica XML", esta es una anotación utilizada por la API del mapeador JAXB. Es porque estamos usando JAXB detrás de escena para mapear XML a Java. Entonces, es una clase auxiliar para esas conversiones JAXB, Java XML. Es autogenerado. No estás escribiendo este. es automatico

Y luego solo controla cómo se asignarán los métodos a los artefactos WSDL. Entonces, métodos web, qué tipo de resultado se produce en su respuesta, qué tipo de solicitud aceptará, sus parámetros web. Estos son los bits que asignan partes del código Java a los bits en un archivo WSDL, específicamente las entradas y salidas de una operación en particular. La operación es "obtener cotización".

JAX-WS Implementation

Create Java implementations of SOAP Services using JAX-WS API.

- Define **Java class** that will provide **Web Service** implementation described by the WSDL file.
- Reference previously described **Java interface** that mapped service operations.
- Describe protocol **Binding Type** (in this case, SOAP1.2 over HTTP).
- Define **Web Methods Implementations** for operations declared by the **Web Service Java Interface**.

```
...
@WebService(serviceName="PriceQuote",
    portName="PriceQuote",
    endpointInterface="com.oracle.xmlns.demos.pricequoteservice.PriceQuote",
    targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService",
    wsdlLocation="WEB-INF/wsdl/PriceQuote/PriceQuoteService.wsdl")
@BindingType(value="http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP/")
public class PriceQuote {
    public float getQuote(Product request) throws QuoteFault {
        ...
    }
}
```



Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

The `BindingType` annotation is used to specify the binding to use for a web service endpoint implementation class. Default is SOAP1.1/HTTP. This example uses SOAP1.2/HTTP binding type.

```
package demos.ws;
import com.oracle.xmlns.demos.pricequoteservice.QuoteFault;
import com.oracle.xmlns.demos.productquote.Product;
import javax.jws.WebService;
import javax.xml.ws.BindingType;
@WebService(serviceName = "PriceQuote",
    portName = "PriceQuote",
    endpointInterface =
"com.oracle.xmlns.demos.pricequoteservice.PriceQuote",
    targetNamespace =
"http://xmlns.oracle.com/demos/PriceQuoteService",
    wsdlLocation = "WEB-INF/wsdl/PriceQuote/PriceQuoteService.wsdl")
@BindingType(value =
"http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP/")
public class PriceQuote {
    public float getQuote(Product request) throws QuoteFault {
        ...
    }
}
```

Ahora la clase de implementación. La clase de implementación podría decir que se trata de un servicio web. Describa qué interfaz de punto final implementa. Puede ser un poco extraño. Ves, normalmente escribirías el código que dice clase, bla, bla, bla, implementa interfaz.

Pero en este caso, está diciendo que la interfaz de punto final es esta o aquella clase. Lo está calificando como un atributo de la anotación del servicio web. Y está describiendo aquí el espacio de nombres y la ubicación del archivo WSDL, dónde están y qué tipo de enlace le gustaría usar. Y básicamente, esta operación es de hecho una implementación detrás, podemos ir a la página anterior, detrás de esa operación.

Y ese "obtener cotización" define el mapeo como solo una interfaz en realidad. Y luego, en este caso, "obtener cotización" contiene un cuerpo real, cualquier lógica que desee poner dentro de ese código. Bueno, qué es lo que se supone que debe hacer la operación de su servicio web.

Create JAVA JAX-WS Client

JAX-WS API is used to implement both SOAP WebServices and Clients.

- Define **Java class** that will represent **Web Service Client**.
- Define **Web Service Port Type** references.

```
/**
 * @WebServiceClient(name="PriceQuote",
 *   targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService",
 *   wsdlLocation="file:wsdl/PriceQuoteService.wsdl")
 */
public class PriceQuote_Service extends Service {
    /**
     * @WebEndpoint(name="PriceQuote")
     */
    public PriceQuote getPriceQuote() {
        return super.getPort(
            new QName("http://xmlns.oracle.com/demos/PriceQuoteService", "PriceQuote"),
            PriceQuote.class);
    }
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Ahora, ¿qué pasa con el cliente? Este es el cliente Java que puede invocar dicho servicio utilizando la anotación `WebServiceClient`. Asigne al mismo archivo WSDL. Ubicación del archivo WSDL. Y representar una pieza de código del lado del servicio en un cliente con este punto final web. Nos permite propagar la llamada del cliente a la operación específica del servidor. En este caso, algún servicio llamado `PriceQuote` que nos gustaría invocar.

Invoke SOAP Service from Java SE Client

Invoke Web Service from Java SE client.

- Acquire **Web Service Port**.
- Prepare **input parameters**.
- Invoke **service operations**.
- Handle **output values** and possible **faults**.

```
/**
 * public static void main(String[] args) {
 *   PriceQuote priceQuotePort = new PriceQuote_Service().getPriceQuote();
 *   Product product = new Product();
 *   product.setId(1);
 *   product.setQuantity(1);
 *   try {
 *     Price quote = priceQuotePort.getPriceQuote(product);
 *   } catch (QuoteFault ex) { ... }
 * }
 */
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Si desea invocar, realice una llamada real, cree `priceQuotePort`, obtenga un puerto. Puerto de servicios web. Forma un documento. En este caso particular, un producto. Presumiblemente, ese producto se asigna a través de JAXB a un esquema XML que utiliza ese servicio. Eso es lo que vas a enviar como tu objeto.

Y páselo a la operación "obtener cotización". Y bueno, sea cual sea el resultado que devuelva, algún resultado de cotización. Ahí tienes Detectar excepciones, si las hay. Las excepciones son básicamente conversiones de sus fallas SOAP a excepciones Java.

Invoke SOAP Service from Java EE Component

Invoke Web Service from Java EE Container Managed component such as Servlet or EJB.

- Inject **Web Service Reference** and acquire **Web Service Port**.
- Prepare **input parameters**.
- Invoke **service operations**.
- Handle **output values**.
- Handle possible **faults**.

```
...
@Stateless
public class OrderManagement {
    @WebServiceRef(
        wsdlLocation="http://localhost:7001/demos/PriceQuote?WSDL")
    private PriceQuoteService service;

    public float getQuote(int id, int quantity)
        throws OrderException {
        PriceQuote priceQuotePort = service.getPriceQuote();
        Product product = new Product();
        product.setId(id);
        product.setQuantity(quantity);
        try {
            float quote = priceQuotePort.getQuote(product);
            return quote;
        } catch (QuoteFault ex) { ... }
    }
}
```



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

When handling web service operation input, output, and fault elements, consider converting JAXB object provided by the web service into your application objects.

Multiple Web Service references can be defined together using `@javax.xml.ws.WebServiceRefs` annotation.

En Java EE, es un poco más simple porque podría usar esta anotación llamada Web Service Ref, que básicamente define la referencia al servicio, y simplemente realiza la inyección directa de un servicio en este bean en particular. Entonces, en este caso, tenemos un bean de sesión sin estado que llama a un servicio web. Ahí tienes El resto de la lógica es la misma que la del cliente Java SE. Esta es solo la forma en que se inyecta. Eso está automatizado por el contenedor.

Summary

In this lesson, you should have learned how to handle communications with SOAP Services, including how to:

- Describe a SOAP Web Service structure
- Create SOAP Web Services using JAX-WS API
- Create SOAP Web Service clients



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Eso es todo. Así que hay dos cosas que cubrimos en este capítulo. Primero, analizamos la estructura de la interfaz del servicio web. Estamos tratando de comprender cómo se presenta el servicio al resto del mundo con WSDL y archivos de esquema. En un caso muy simple, todo lo que necesita hacer es simplemente decir "servicio web de la aplicación" y le generará WSDL y esquemas fuera de una clase. Podrías tomar ese enfoque.

Por supuesto, si desea tener más control sobre cómo exactamente sus clases de Java se asignan a WSDL y esquemas, deberá usar todas estas otras anotaciones. Método web para designar qué operación se asigna a WSDL, resultados web para designar cómo se asigna el tipo de operación de retorno, parámetro web para designar cómo se asignan los parámetros. Y controle la forma en que el código Java se asigna a WSDL y esquemas.

En un ejercicio práctico, creará un servicio web llamado Cotización de precios que básicamente va a la misma base de datos de productos y devuelve cotizaciones de productos. Y luego consumir eso del cliente. Escriba un cliente Java que llamará a ese servicio.