Wsdl

WSDL (Web Services Description Language) es un lenguaje basado en XML utilizado para describir la funcionalidad de un servicio Web. Permite comprender la interfaz del servicio Web, indicando cómo se debe llamar al servicio, qué parámetros espera, y qué estructuras de datos devuelve. Básicamente se trata del contrato entre quien produce el servicio web y quien lo consume.

Los elementos mas importantes de un WSDL son:

- types: define los tipos de datos que se intercambiarán en el mensaje. La definición de tipos puede verse como las definiciones Java de clase, con variables que pueden ser tipos primitivos o referencias a otras clases u objetos.
- message: define los distintos mensajes que se intercambiarán en la llamada al servicio web. Se definen los mensajes de entrada y salida para cada operación que ofrezca el servicio.
- portType: contiene una colección de una o más operaciones. Para cada una, indica cuáles son los mensajes de entrada y salid.
- binding: define el protocolo de red y el formato de los datos para las operaciones de un portType. Los bindings son definiciones concretas de los portTypes. Un portType puede tener múltiples bindings asociados.

Top Down y Bottom Up

Top Down (de arriba a abajo) y Bottom Up (de abajo a arriba) son estrategias utilizadas para el desarrollo de software en general.

En el diseño top-down se formula un modelo del sistema, sin especificar detalles. Cada parte nueva es redefinida cada vez con mayor detalle, hasta que la especificación completa es lo suficientemente detallada para validar el modelo.

En el diseño bottom-up las partes individuales se diseñan con detalle y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo.

Aplicando estos métodos a la construcción de servicios web en un enfoque Bottom Up primero se define la lógica del servicio web y a partir del mismo se construyen las interfaces. Primero se escribe el código del servicio y luego se genera el WSDL utilizando el código escrito.

Como su nombre lo indica Top Down es el camino inverso al del enfoque Bottom Up. Primero se escribe la definición del servicio. El WSDL se crea primero, luego el servicio es escrito a partir de las definiciones del WSDL. Existen herramientas que utilizando ese WSDL generan un esqueleto del código necesario para luego escribir la lógica del servicio a desarrollar, una de esas herramientas es wsimport.

Wsimport

Wsimport es una de las herramientas que se encuentran dentro de la JDK para automatizar la generación de código desde un WSDL.

Sintaxis

Existen pequeñas diferencias en el uso de este comando en las versiones para sistemas operativos Windows y Unix. En windows la herramienta está en \bin\wsimport.bat y en linux se puede encontrar en /bin/wsimport.sh.

La sintaxis en ambos sistemas es la misma:

wsimport [options] <wsdl>

<wsdl> indica la url para acceder al WSDL del servicio web para el que vamos a generar el cliente. Esta url puede ser tanto un recurso local, como un recurso obtenido mediante protocolo http.

[options] es el apartado donde se pueden incluir diferentes opciones que variarán el comportamiento de la herramienta. A continuación se presenta una lista completa de estas opciones:

- -d <directory> : Indica el directorio de salida donde se dejan las clases compiladas. Si no se utiliza esta opción las clases compiladas se dejarán en el mismo directorio desde el que se llama a wsimport.
- -b <path> : Añade ficheros XSD adicionales que se puedan necesitar en los binding jaxws/jaxb del servicio web.
- -B <jaxbOption>
- -catalog <file> : Especifica un fichero de catálogo que resuelve las referencias a entidades externas. Los formatos de catálogo soportados son: TR9401, XCatalog y OASIS XML Catalog.
- -extension : Permite extensiones de terceros. Esta funcionalidad no está soportada por la especificación, por lo que el cliente generado puede no ser portable o permitir la interoperabilidad entre plataformas.
- -help : Muestra una ayuda con el listado de las opciones.
- -httpproxy:<host>:<port> : Si para acceder a la URL del WSDL que queremos generar hay que viajar a través de un proxy, con esta opción podremos indicar su configuración. Si no se rellena el puerto, por defecto será el 8080.
- -keep : Si se incluye esta opción los fuentes que generan los compilados del cliente no se borran.
- -p : Especifica el paquete java de las clases del cliente generado. Si se indica esta opción no se tendrán en cuenta; ni el nombre de paquete que puede incluirse en el wsdl, ni el nombre de paquete por defecto que se genera cuando no se indica esta opción.
- -s <directory> : Especifica un directorio donde se guardan los ficheros de código fuente generados.
- -verbose : Muestra los mensajes del compilador indicando las tareas que está realizando.
- -version : Muestra un mensaje informativo con la versión de la implementación de referencia que se está utilizando.
- -wsdllocation <location>
- -target : Genera el código para la versión JAX-WS indicada. La versión 2.0 genera código compatible con la especificación JAX-WS 2.0.
- -quiet: Elimina cualquier salida que se pueda generar. Útil para generaciones automatizadas de clientes.

Clases generadas

Wsimport genera las clases para poder invocar a las operaciones del servicio correctamente. Las clases generadas siguen el siguiente criterio:

- Clase PortType. Una clase que lleva el mismo nombre que el atributo name del elemento porttype del wsdl y contiene un método por cada operación definida con los elementos operation.
- Clase Service. Una clase que lleva el mismo nombre que el atributo name del elemento service del wsdl. Esta clase accede al servicio web y permite instanciar la clase PortType.
- Por cada operación definida en el portType

- Tantas clases como sean necesarias para rellenar los Input
- Tantas clases como sean necesarias para devolver el resultado de la operación

- Clase ObjectFactory. Esta clase facilita la instanciación interna de las clases input y response.
- Clase package-info. Anota el paquete java para que los objetos generados a partir del xsd del wsdl estén correctamente ubicados.

Anotaciones

Podemos especificar la forma en la que se crea el servicio mediante diferentes anotaciones. Las principales anotaciones disponibles son:

@WebService	Indica la clase que define un servicio web. Se pueden especificar como parámetros los nombres del servicio (serviceName), del componente Port (portName), del SEI del servicio (name), de su espacio de nombres (targetNamespace), y de la ubicación del WSDL (wsdlLocation), que figurarán en el documento WSDL del servicio:
	`@WebService(name="ConversionPortType",`
	`  serviceName="ConversionService",`
	`  portName="ConversionPort",`
	`  targetNamespace="http://jtech.ua.es",`
	`  wsdlLocation="resources/wsdl/")`
	Cuando se consume un WS, define de forma abstracta un servicio web, las operaciones que puede llevar a cabo y los mensajes involucrados en cada operación.

@WebMethod	Indica que un determinado método debe ser publicado como operación del servicio. Si no se indica para ningún método, se considerará que deben ser publicados todos los métodos públicos. Si no, sólo se publicarán los métodos indicados. Además, de forma opcional se puede indicar como parámetro el nombre con el que queramos que aparezca la operación en el documento WSDL:
	`@WebMethod(operationName="eurosAptas")`
	`public int euro2ptas(double euros) {`
	`  ...`
	`}`

@Oneway	Indica que la llamada a la operación no debe esperar ninguna respuesta. Esto sólo lo podremos hacer con métodos que devuelvan void. Por ejemplo:
	`@Oneway()`
	`@WebMethod()`
	`public void publicarMensaje(String mensaje) {`
	`  ...`
	`}`

@WebParam	Permite indicar el nombre que recibirán los parámetros en el fichero WSDL:
	`@WebMethod(operationName="eurosAptas")`
	`public int euro2ptas(`
	` @WebParam(name="CantidadEuros",`
	`  targetNamespace="http://jtech.ua.es")`
	` double euros) {`
	`  ...`
	`}`

@WebResult	Permite indicar el nombre que recibirá el mensaje de respuesta en el fichero WSDL:
	`@WebMethod(operationName="eurosAptas")`
	`@WebResult(name="ResultadoPtas",`
	`  targetNamespace="http://jtech.ua.es")`
	`public int euro2ptas(double euros) {`
	`  ...`
	`}`

@WebFault	Se utiliza para anotar excepciones Java. Cuando utilizamos esta anotación en una excepción estamos indicando que cuando sea lanzada por una operación del servicio web debe generar un mensaje SOAP de respuesta con un SOAP Fault que nos indique el error producido. En el lado del cliente la clase con dicha excepción se habrá generado en el stubpara el acceso al servicio, y al recibir el mensaje SOAP con el error el stub lanzará la excepción correspondiente. Es decir, para el desarrollador será como si la excepción saltase directamente desde el servicio hasta el cliente.
	@WebFault
	public class ConversionFaultException extends Exception {
	public ConversionFaultException(String msg) {
	super(msg);
	}
	}

@WebServiceClient	Representa el punto de partida del servicio web para consumir. Esta es la clase a instanciar para obtener una implementación del WS. Una vez instanciada esta clase, es posible obtener el “port type”, llamando al método que está anotado con @WebEndpoint. El port type no es más que una implementación del servicio definido en @WebService.

Wsgen

Wsgen es una herramienta por línea de comando disponible en la JDK (Java development Kit) que permite generar todos los artefactos portables utilizados en los servicios web JAX-WS. Wsgen lee la clase de la implementación de un web service endpoint (SEI) y genera todos los artefactos requeridos para el deploy e invocación de un Web Service.

Existen dos casos de uso comunes para utilizar wsgen:

1. Generar JAX-WS portable artifacts (archivos Java) para el deploy de un web service.
2. Generar los archivos WSDL y XSD para deploy o testing de un web service client.

Ejemplos:

wsgen -verbose -keep -cp <paquete.clase del SEI>

Esto genera tanto los archivos .class como los .java gracias al parámetro -keep.

wsgen -verbose -keep -cp <paquete.clase del SEI> -wsdl

Esto genera además de los archivos .class y los .java, los archivos WSDL y XSD gracias al parámetro -wsdl.

Xjc

Xjc es una herramienta por línea de comando disponible en la JDK (Java development Kit) que permite generar clases Java con anotaciones de JAXB a partir de un archivo XSD. Esto puede ser útil si contamos con el XSD de

los mensajes que nuestro web service debe cumplir. Un caso de uso común sería cuando tenemos que construir un web service a partir de la definición de los XML que deberían viajar, en ese caso se puede utilizar alguna herramienta para obtener el XSD a partir del XML y con xjc a partir del XSD obtener las clases Java necesarias con las anotaciones JAXB que validen correctamente el XSD.

Ejemplo:

```
xjc <archivo.xsd>
```