

INSTITUTO TECNOLOGICO DE NUEVO LEON

ALUMNA:

EDITH MENDOZA CONTRERAS

16480063

MATERIA:

LENGUAJES Y AUTOMATAS 1

INVESTIGACION DE JAX Y EJEMPLO EN JAVA.

JAX –WS JAVA

Es una interfaz de programación de aplicaciones (API) de java en Extensible markup lenguaje (XML) para la creación de servicios web (WS). Es parte de la plataforma java EE de sun Microsystems. Al igual que las otras API de java EE, JAX-WS utiliza anotaciones, introducidas en java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de servicios web.

La implementación de referencia de JAX-WS se desarrolla como un proyecto de código abierto y forma parte del proyecto GlassFish, un servidor de aplicaciones java EE de código abierto. Se llama JAX-WS RI y se dice que es la implementación de calidad de producción (contrariamente a la implementación de referencia antigua que era una prueba de concepto). Esta implementación de referencia es ahora parte de la distribución es ahora parte de la distribución metro.

JAX-WS también es uno de los fundamentos de WSIT.

JAX-WS es un modelo de programación que simplifica el desarrollo de aplicaciones mediante el soporte de un modelo basado en anotaciones estándar para desarrollar clientes y aplicaciones de servicio web.

En JAX-WS, una invocación de operación de servicio web se representa mediante un protocolo basado en XML, como SOAP. La especificación SOAP define la estructura del sobre, las reglas de codificación y las convenciones para representar las invocaciones y respuestas de servicios web. Estas llamadas y respuestas se transmiten como mensajes SOAP (archivos XML) a través de HTTP.

Aunque los mensajes SOAP son complejos, la API JAX-WS oculta esta complejidad al desarrollador de la aplicación. En el lado del servidor, el desarrollador especifica las operaciones del servicio web definiendo métodos en una interfaz escrita en el lenguaje de programación Java.

Interoperabilidad de los WS: Metro y JAX-WS

La interoperabilidad de los servicios Web constituye una iniciativa de Sun y Microsoft. El principal objetivo es proporcionar productos que sean capaces de interpolar a través de diferentes plataformas.

JAX-WS define un modelo de programación en el que se realiza un mapeado de un documento WSDL a Java. Dicho mapeado proporciona una factoría (Service) para seleccionar qué Port (agregado en el servicio) desea usar el cliente. Como veremos más adelante, la herramienta JAX-WS que proporciona las clases necesarias en la parte del cliente para poder acceder a un servicio web se denomina wsimport.

. Tipos de acceso para invocar servicios Web JAX-WS

Tenemos dos formas diferentes de invocar un Servicio Web utilizando JAX-WS (llamaremos servicios web JAX-WS a aquellos servicios que utilizan la librería JAX-WS).

Creación de un stub estático: Consiste en generar una capa de stub por debajo del cliente de forma automática. Dicho stub implementará la misma interfaz que el servicio, lo cual nos permitirá desde nuestro cliente acceder al Servicio Web a través del stub tal y como si estuviéramos accediendo directamente al servicio.

Utilización de la Interfaz de Invocación Dinámica (DII): Esta forma de acceso nos permitirá hacer llamadas a procedimientos de nuestro Servicio Web de forma dinámica, sin crear un stub para ello.

Invocación de servicios web JAX-WS con JDK 1.6

Para crear un cliente en JDK 1.6 (o con JAX-WS en versiones anteriores de JDK) utilizaremos la herramienta wsimport, que tomará como entrada el documento WSDL del servicio al que queremos acceder y producirá un conjunto de clases Java que nos permitirán acceder al servicio. Esta herramienta se puede invocar desde línea de comando: wsimport -s -d –p

Invocación de servicios web JAX-WS desde una clase Java con Maven

EJEMPLO

Generar un web service con metro.

62.983 activation.jar

291.817 FastInfoset.jar

21.839 gmbal-api-only.jar

82.265 http.jar

104.554 jaxb-api.jar

876.738 jaxb-impl.jar

3.105.074 jaxb-xjc.jar

54.341 jaxws-api.jar

1.484.080 jaxws-rt.jar

520.641 jaxws-tools.jar

23.346 jsr173\_api.jar

7.993 jsr181-api.jar

6.165 jsr250-api.jar

41.429 management-api.jar

38.772 mimepull.jar

156.212 policy.jar

68.177 resolver.jar

18.774 saaj-api.jar

288.529 saaj-impl.jar

11.001 stax-ex.jar

59.771 streambuffer.jar

505.825 woodstox.jar

Una vez hecho esto, sólo nos queda escribir nuestro código. No tenemos más que escribir la clase que será nuestro *Web Service* con sus métodos, que serán los accesibles desde la Web. Para indicar que la clase es un *Web Service* sólo tendremos que ponerle la anotación *@WebService* y a los métodos que queramos que sea accesibles la anotación *@WebMethod*. La clase sería la siguiente

package com.chuidiang.ejemplos.jax\_ws;

import javax.jws.WebMethod;

import javax.jws.WebService;

@WebService

public class UnWebService {

@WebMethod

public float suma(float a, float b, UnDato c) {

return a + b;

}

}

Podríamos compilar y generar un fichero .war con esta clase para desplegarla en un [Tomcat](http://chuwiki.chuidiang.org/index.php?title=Instalaci%C3%B3n_de_Tomcat_en_Windows) o similar, pero podemos directamente ponerle un *main()* para hacerla ejecutable. Para hacer pública la clase *UnWebService* como *Web Service* podemos usar la clase *EndPoint* que viene con *jax-ws*. El código quedaría así

package com.chuidiang.ejemplos.jax\_ws;

import javax.jws.WebMethod;

import javax.jws.WebService;

import javax.xml.ws.Endpoint;

@WebService

public class UnWebService {

@WebMethod

public float suma(float a, float b, UnDato c) {

return a + b;

}

public static void main(String[] args) {

Endpoint.publish("http://localhost:8080/UnWebService", new UnWebService());

}

}