Desarrollo de Servicios Web 1

Tema Nº2:SERVICIOS SOAP

Indicador de logro Nº2:Aplica los fundamentos XML y JSON a través del Web Service SOAP mediante operaciones básicas.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº2:**

SERVICIOS SOAP

**Subtema 2.1:**

Primero llegó SOAP, luego REST

**MARCO TEÓRICO**

La primera introducción de los webs services en el mundo de Internet vino de la mano de SOAP. SOAP es un protocolo que define cómo deben de realizarse las comunicaciones entre máquinas. SOAP usa XML como lenguaje de intercambio de datos con una estructura compleja que es capaz de albergar todo tipo de datos sobre la solicitud o respuesta generada.

REST usa el propio protocolo HTTP para la comunicación entre máquinas. HTTP es ampliamente soportado por todos los sistemas y de hecho para la transferencia de datos en la web se usa HTTP.

REST se caracteriza por no tener estado. Es decir, el servidor no es capaz de recordar el estado de la anterior solicitud REST que pudo, o no, hacer un cliente. Por ello, el cliente tiene que enviar en cada solicitud todo el estado de su sesión, lo que se suele hacer mediante un token que le «ayude a recordar» al servidor.

Mencionamos esas dos características porque es justamente lo permite que la implementación de REST sea realmente fácil y se haya popularizado tanto el tipo de servicio web que se conoce como el nombre de API REST.

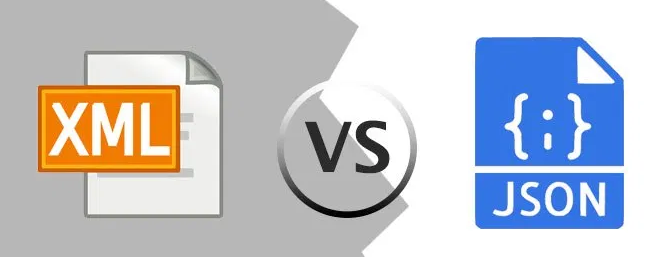
REST tiene a simplificar las cosas y en esa misma línea se suele usar un lenguaje diferente para representación de los datos, el lenguaje JSON. Hoy REST y JSON se han convertido en la opción más sencilla y por tanto más recomendable para implementar un servicio web.

**XML vs JSON**

Para los sistemas de comunicación entre máquinas se requiere una serie de características, básicamente marcadas porque las máquinas implicadas en la comunicación pueden tener sistemas muy diferentes. Pueden usar lenguajes de programación o bases de datos diferentes y hasta los sistemas operativos suelen ser distintos entre clientes y servidores. Por ello, para la comunicación de los datos se usa básicamente lenguajes escritos en archivos de texto plano.

Entre los lenguajes más usados para la comunicación de los datos en los servicios web, encontramos XML y JSON.

* XML está basado en etiquetas, como HTML. Es más tradicional pero también es un lenguaje más avanzado, que presenta diversas utilidades para su extensión, validación de la información y sintaxis de los datos, etc.
* JSON es un lenguaje más nuevo, basado en sintaxis Javascript. Generalmente, es más ligero y requiere mucho mejor carga del servidor para su procesamiento.



**SOAP Request**

<?xml version=*"1.0"*?>

<soap:Envelope

xmlns:soap=*"http://www.w3.org/2001/12/soap-envelope"*

soap:encodingStyle=*"http://www.w3.org/2001/12/soapencoding"*>

<soap:Body><m:ObtenerPrecio

xmlns:m=*"http://www.w3schools.com/prices"*>

<m:Elemento>Naranjas</m:Elemento>

</m:ObtenerPrecio></soap:Body>

</soap:Envelope>

**SOAP Response**

<?xml version=*"1.0"*?>

<soap:Envelope

xmlns:soap=*"http://www.w3.org/2001/12/soap-envelope"*

soap:encodingStyle=*"http://www.w3.org/2001/12/soapencoding"*>

<soap:Body><m:ObtenerPrecioRespuesta

xmlns:m=*"http://www.w3schools.com/prices"*>

<m:Precio>2.20</m:Precio>

</m:ObtenerPrecioRespuesta></soap:Body>

</soap:Envelope>

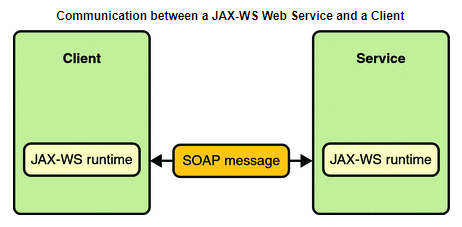
**Subtema 2.2:**

JAX-WS

**MARCO TEÓRICO**

JAX-WS (Java™ API for XML-Based Web Services) es la siguiente generación del modelo de programación de servicios web que complementa la infraestructura que proporciona el modelo de programación JAX-RPC (Java API for XML-based RPC). Con JAX-WS, se simplifica el desarrollo de servicios web y clientes con una mayor independencia de la plataforma para las aplicaciones Java mediante el uso de anotaciones Java y proxies dinámicos.

JAX-WS proporciona el soporte para anotar clases Java con metadatos para indicar que la clase Java es un servicio web. JAX-WS da soporte al uso de anotaciones basándose en la especificación del recurso de metadatos del lenguaje de programación Java (JSR 175), la especificación de metadatos de servicios web para la plataforma Java (JSR 181) y las anotaciones definidas por la especificación JAX-WS 2.2. Utilizando las anotaciones del origen Java y la clase Java se simplifica el desarrollo de los servicios web. Utilice las anotaciones para definir la información que se suele especificar en los archivos del descriptor de despliegue, los archivos WSDL o en la correlación de metadatos de archivos XML y WSDL con los artefactos de origen.



**Anotaciones**

|  |  |
| --- | --- |
| @WebService | Indica que la clase 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/") |
| @SOAPBinding | Permite especificar el estilo y la codificación de los mensajes SOAP utilizados para invocar el servicio. Por ejemplo:  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT,  use=SOAPBinding.Use.LITERAL,  parameterStyle=SOAPBinding.ParameterStyle.WRAPPED) |
| @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 stub para 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);  }  } |

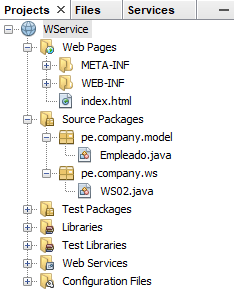
**1. MATERIALES**

Para la experiencia a realizar se requiere lo siguiente:

1. EQUIPO

Se requiere tener un equipo de cómputo (PC o laptop).

1. HERRAMIENTAS

* Java Developer Kit (JDK) version 8.x
* NetBeans 8.2
* Apache Tomcat 9.0.12
* Jax-ws/metro
* SoapUI 5.5

**2. PROCEDIMIENTO**

Se tiene el proyecto Web:

Clase Empleado:

package pe.company.model;

import java.util.Date;

public class Empleado

{

private Integer id\_empleado;

private String nombre;

private Date fecha\_contrato;

private String profesion;

private Double sueldo;

public Empleado() {}

public Empleado(Integer id\_empleado, String nombre, Date fecha\_contrato,

String profesion, Double sueldo) {

this.id\_empleado = id\_empleado;

this.nombre = nombre;

this.fecha\_contrato = fecha\_contrato;

this.profesion = profesion;

this.sueldo = sueldo;

}

//Getters and Setters

}

Web Service WS02:

package pe.company.ws;

import java.util.\*;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

import pe.company.model.Empleado;

@WebService(serviceName = "WS02")

public class WS02

{

//colección de empleados en memoria

private static final Collection<Empleado> *itemsEmpleado*=new ArrayList();

static

{

//empleados

Empleado empleado1=new Empleado(1021,

"Armando",new GregorianCalendar(2014,03,10).getTime(),"Contador",4000.0);

Empleado empleado2=new Empleado(1022,

"Margarita",new GregorianCalendar(2018,9,20).getTime(),"Ingeniero",6000.0);

Empleado empleado3=new Empleado(1023,

"Basilio",new GregorianCalendar(2015,1,8).getTime(),"Arquitecto",5000.0);

Empleado empleado4=new Empleado(1024,

"Verena",new GregorianCalendar(2019,11,21).getTime(),"Obrero",2000.0);

//guardar empleados en colección

*itemsEmpleado*.add(empleado1);

*itemsEmpleado*.add(empleado2);

*itemsEmpleado*.add(empleado3);

*itemsEmpleado*.add(empleado4);

}

//operación del web service

@WebMethod(operationName = "baraja")

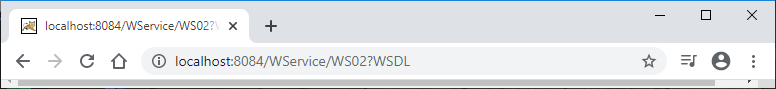
public Collection<Empleado> listarEmpleados(@WebParam(name = "codigo") String codigo) {

return *itemsEmpleado*;

}

}

Al ejecutar el proyecto Web, se tendrá el WSDL:



<definitions

xmlns:wsu=*"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"*

xmlns:wsp=*"http://www.w3.org/ns/ws-policy"*

xmlns:wsp1\_2=*"http://schemas.xmlsoap.org/ws/2004/09/policy"*

xmlns:wsam=*"http://www.w3.org/2007/05/addressing/metadata"*

xmlns:soap=*"http://schemas.xmlsoap.org/wsdl/soap/"*

xmlns:tns=*"http://ws.company.pe/"*

xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*

xmlns=*"http://schemas.xmlsoap.org/wsdl/"*

targetNamespace=*"http://ws.company.pe/"* name=*"WS02"*>

<types>

<xsd:schema>

<xsd:import namespace=*"http://ws.company.pe/"*

schemaLocation=*"http://localhost:8084/WService/WS02?xsd=1"* />

</xsd:schema>

</types>

<message name=*"baraja"*>

<part name=*"parameters"* element=*"tns:baraja"* />

</message>

<message name=*"barajaResponse"*>

<part name=*"parameters"* element=*"tns:barajaResponse"* />

</message>

<portType name=*"WS02"*>

<operation name=*"baraja"*>

<input wsam:Action=*"http://ws.company.pe/WS02/barajaRequest"* message=*"tns:baraja"* />

<output wsam:Action=*"http://ws.company.pe/WS02/barajaResponse"*

message=*"tns:barajaResponse"* />

</operation>

</portType>

<binding name=*"WS02PortBinding"* type=*"tns:WS02"*>

<soap:binding

transport=*"http://schemas.xmlsoap.org/soap/http"* style=*"document"* />

<operation name=*"baraja"*>

<soap:operation soapAction=*""* />

<input> <soap:body use=*"literal"* /> </input>

<output> <soap:body use=*"literal"* /> </output>

</operation>

</binding>

<service name=*"WS02"*>

<port name=*"WS02Port"* binding=*"tns:WS02PortBinding"*>

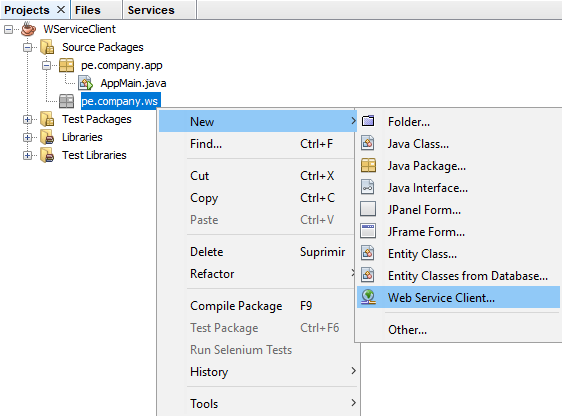
<soap:address location=*"http://localhost:8084/WService/WS02"* />

</port>

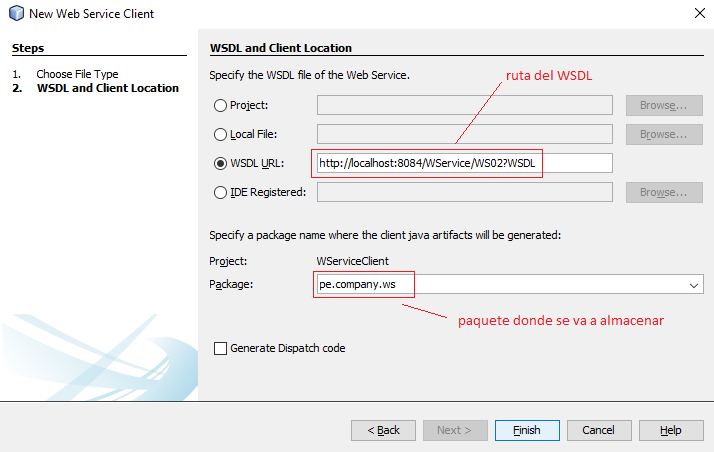
</service>

</definitions>

Se pasa a crear un Web Service Client (App Console) para consumir el servicio:

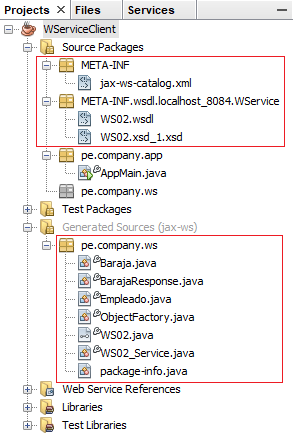


Luego hacemos:



Finish, y se espera hasta que el IDE pueda clonar el WSDL:

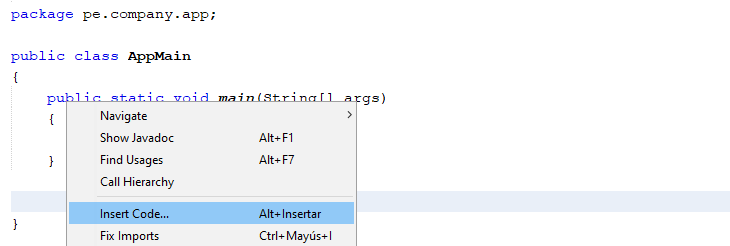
Se observa:

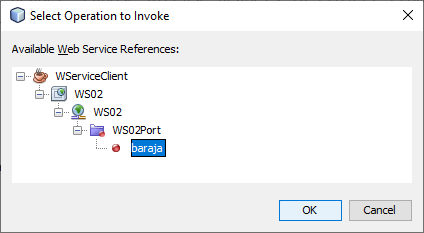


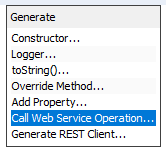
El WSDL

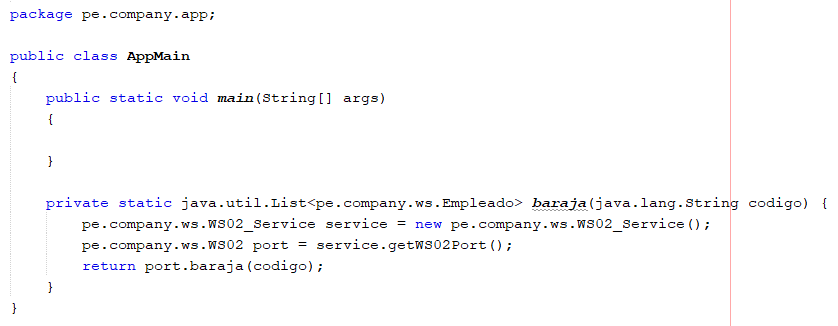
Clases Java que consumirán el servicio

En AppMain se va a llamar a una operación del Web Service:









A continuación, se llamará el servicio desde esta App Client:

package pe.company.app;

import java.util.Collection;

import pe.company.ws.Empleado;

public class AppMain

{

public static void main(String[] args)

{

//obteniedo empleados por código

Collection<Empleado> itemsEmpleado=*baraja*("1021");

//imprimiendo listando empleados

for(Empleado item:itemsEmpleado)

{

System.*out*.println("ID Empleado: "+item.getIdEmpleado());

System.*out*.println("Nombre: "+item.getNombre());

System.*out*.println("Fecha de contrato: "+item.getFechaContrato());

System.*out*.println("Profesión: "+item.getProfesion());

System.*out*.println("Sueldo: S/."+item.getFechaContrato()+"\n");

}

}

private static java.util.List<pe.company.ws.Empleado> baraja(java.lang.String codigo) {

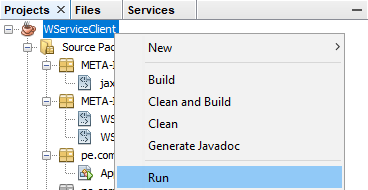
pe.company.ws.WS02\_Service service = new pe.company.ws.WS02\_Service();

pe.company.ws.WS02 port = service.getWS02Port();

return port.baraja(codigo);

}

}



Se observa lo que está devolviendo el servicio:

run:

ID Empleado: 1021

Nombre: Armando

Fecha de contrato: 2014-04-10T00:00:00-05:00

Profesión: Contador

Sueldo: S/.2014-04-10T00:00:00-05:00

ID Empleado: 1022

Nombre: Margarita

Fecha de contrato: 2018-10-20T00:00:00-05:00

Profesión: Ingeniero

Sueldo: S/.2018-10-20T00:00:00-05:00

ID Empleado: 1023

Nombre: Basilio

Fecha de contrato: 2015-02-08T00:00:00-05:00

Profesión: Arquitecto

Sueldo: S/.2015-02-08T00:00:00-05:00

ID Empleado: 1024

Nombre: Verena

Fecha de contrato: 2019-12-21T00:00:00-05:00

Profesión: Obrero

Sueldo: S/.2019-12-21T00:00:00-05:00

BUILD SUCCESSFUL (total time: 10 seconds)

Nota: Esta App Console puede ser una App Android, App Desktop, Placa Arduino, etc.

**ACTIVIDAD VIRTUAL:**

1. **CUESTIONARIO TÉCNICO**

Revisa y analiza el tema desarrollado en la presente sesión, luego responde las siguientes preguntas propuestas:

* ¿El transporte de datos en XML a que tecnología pertenece, dé una breve explicación?
* ¿El transporte de datos en JSON a que tecnología pertenece, dé una breve explicación?
* ¿SOAP Request y SOAP Response en que se diferencia?
* ¿Los archivos WSDL para que sirve dentro de los servicios JAX-WS?
* Realice un Web Service mediante un código y mostrar sus datos personales.

1. **ENLACES DE REFERENCIA**

* https://www.arsys.es/blog/programacion/web-services-desarrollo/
* https://www.ibm.com/docs/es/was/9.0.5?topic=services-jax-ws
* http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion02-apuntes.html

1. **CONCLUSIONES DE LA EXPERIENCIA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_